NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

INTERACTIVE MULTIMEDIA
FOR
CLASSROOM AND WEB USE

by

Darren S. Harvey

March 1999

Thesis Advisor:	Jon T. Butler
Second Reader:	Herschel H. Loomis

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** March 1999	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE Interactive Multimedia for Classroom and Web Use	**5. FUNDING NUMBERS**
6. AUTHOR(S) Harvey, Darren S.	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not Reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution unlimited.	**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(maximum 200 words)*** Today's advances in technology have provided new tools that allow us to be more creative. The focus of this research is on interactive lesson plans and a web-based tutorial for a graduate level computer architecture class. It contains recommendations for software selection and development methods. In addition, this thesis examines the learning process and describes the necessary ingredients of developing an effective multimedia-based production. Instructional system design (ISD) principles are examined in depth and used as a basis to design the interactive tutorial. Finally, the study discusses the feasibility of developing these productions considering the cost and time investment.

14. SUBJECT TERMS Computer-based learning, multimedia, tutorials, lesson plan development	**15. NUMBER OF PAGES** 171
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFI- CATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std.

THIS PAGE INTENTIONALLY LEFT BLANK

INTERACTIVE MULTIMEDIA FOR CLASSROOM AND WEB USE

Darren S. Harvey
Lieutenant, United States Navy
B.S., University of South Alabama, 1991

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 1999

Author: [redacted]
Darren S. Harvey

Approved by: [redacted]
Jon T. Butler, Thesis Advisor

[redacted]
Herschel H. Loomis, Second Reader

[redacted]
Jeffrey B. Knorr, Chairman
Department of Electrical and Computer Engineering

ABSTRACT

Today's advances in technology have provided new tools that allow us to be more creative. The focus of this research is on interactive lesson plans and a web-based tutorial for a graduate level computer architecture class. It contains recommendations for software selection and development methods. In addition, this thesis examines the learning process and describes the necessary ingredients of developing an effective multimedia-based production. Instructional system design (ISD) principles are examined in depth and used as a basis to design the interactive tutorial. Finally, the study discusses the feasibility of developing these productions considering the cost and time investment.

TABLE OF CONTENTS

I. INTRODUCTION

A. AREA OF RESEARCH

Today's advances in technology have provided new tools that allow us to be more creative, innovative and better problem solvers. A question that has challenged academia, industry and society in general, is how can we attain maximum efficiency in training and education. This question inspires the examination of non-traditional teaching methods. Examples of traditional methods are strictly book-based reading and typical lecturing. These are time-proven methods which have produced good results. But the question still remains, is this the most efficient manner to educate? In a world where technologies are changing so rapidly, non-traditional methods of education and training offer a unique opportunity.

The focus of the research described in this thesis is to develop interactive lesson plans and a web-based tutorial for the graduate level computer architecture class, EC 3840 taught in the Electrical Engineering Department at the Naval Postgraduate School (NPS). This is an application of the concept of computer-mediated learning. Gayeski, describes computer-mediated learning as the use of interactive, microprocessor-based digital media to aid individuals in learning and applying new concepts and skills. [Ref. 1]

The multimedia productions developed from this thesis were designed using commercial off-the-shelf software. The software was evaluated based on price, interface and ease-of-use. The productions consist of audio, video, animations and 3-D graphics. The productions are targeted to address typically difficult concepts such as pipelining and register-transfer notation. Another accomplishment of the research was the development of a series of steps and considerations to aid in the development and maintenance of the productions. Additionally, the economic feasibility and practicality of these education aids were considered.

B. RESEARCH QUESTIONS

- Using commercial off-the-shelf multimedia development software technology, can we develop an effective multimedia production, as a supplement to classes?
- Can these systems be used in the classroom?
- Is the development of such productions cost and time effective?
- Can a guide or protocol be developed that would make the creation of the productions more systematic and allow for changes to be incorporated by different designers in efficient manner?
- What particular training is required and will the skills learned be beneficial to Navy personnel?

C. ACCOMPLISHMENTS OF THE STUDY

A multimedia-based tutorial and a five-stage pipelined processor were designed as a result of the study. A guide for the development and revision of multimedia-based productions has been identified and explained in this study. The study also addresses software selection and contains my recommendations and comments on all the software used to create the productions.

D. FINDINGS FROM THE STUDY

This research lays the groundwork for interactive multimedia web-based tutorials, as well as multimedia content lesson plans for EC 3840. It provides a guide for the development and maintenance of future productions. The study places great significance in implementing a standardized development system that addresses the needs of the learner and the developer. As will be discussed later, the technologies used to implement multimedia productions become irrelevant if the framework used to present the content lacks the principles of certain instructional precepts. That is, the multimedia technology only provides a platform for implementing new non-traditional teaching techniques. The techniques themselves must be sound.

Chapters II to IV provide the explanations of the fundamental processes required for designing technology based applications. These initial chapters address the learning process, descriptions of instructional architectures that have proven to be effective in teaching, and descriptions of the types of media commonly used in multimedia productions. It is suggested that Chapter IV, The Development Process, be considered the most important chapter of the study. Chapter IV provides a system that can be used to develop, maintain and revise technology based projects. The principles discussed are technology independent and are based on many of the concepts found in systems engineering.

Chapter II, Using Multimedia For Instruction, is an overview of some important principles dealing with the development of multimedia-based learning. The concept of multimedia is defined and the relevance of its use is discussed. The importance of understanding and implementing the proper instructional architectures for optimal efficiency in learning is discussed.

Chapter III, Cost and Time Justification, addresses some of considerations that need to be examined when considering multimedia-based learning.

Chapter IV, The Development Process, introduces the Instructional System Design (ISD) process, which is the basis for multimedia production design. This chapter gives a brief overview and explanation of the phases of the ISD process.

Chapter V, Evaluation, gives a brief description of some evaluation techniques used in developing multimedia-based productions.

Chapter VI, Software Selection, discusses the software requirements, specific applications and the criteria and logic behind the selection of particular software packages. Recommendations on application software products are also included.

Chapter VII, Web-based Tutorial Development, examines the process of the design and development of the multimedia-based learning environment as it relates to the EC 3840 tutorials and it's presentation via the Internet.

Chapter VIII, Testing and Results, deals with actual implementation and publishing the final multimedia-based product to the Web. Problems encountered and recommendations for the efficient transition from development to publishing are also addressed.

Chapter IX contains the conclusions and final recommendations.

II. USING MULTIMEDIA FOR INSTRUCTION

A. BACKGROUND

The two major divisions of multimedia are

1. **Multimedia in the Information Field**

2. **Digital Multimedia**

Fluckinger [Ref. 2] defines the term multimedia in the information field as:

> Multimedia in the general information field means "multiple intermediaries" between the source and sink of information of "multiple means" by which information is stored, transmitted, presented, or perceived.

He continues by defining digital multimedia:

> Digital multimedia is the field concerned with the computer-controlled integration of text, graphics, still and moving images, animation sounds, and any other medium where every type of information can be represented, stored, transmitted, and processed digitally. [Ref. 2]

In the development of the computer-based learning (CBL) environment, both the informational field and the digital multimedia aspects are relevant. This is because we are concerned with the storing and processing of information, as well as its presentation.

Figure 1 shows that multimedia can be described as six different types of elements; text, graphics, images, moving graphics (animation), moving images (video) and sound. [Ref.

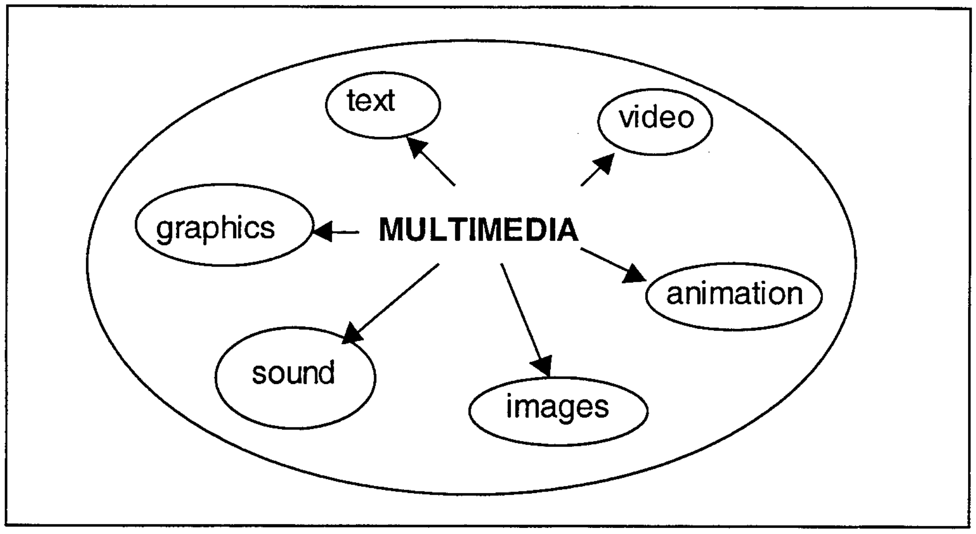


Figure 1. Multimedia components

2] The overall intent of all of these elements is to widen the range of presentation by stimulating specific senses, e.g. sight and sound. This is one of the primary advantages of using multimedia as a learning tool.

Fluckinger continues his discussion of multimedia by introducing four requirements of multimedia systems:

- Multimedia systems must be computer controlled.
- Systems must be integrated.
- The information handled must be represented digitally.
- The interface to the final user must permit interactivity.

B. WHY MULTIMEDIA COMPUTER-BASED TRAINING?

An important question is, "How effective is multimedia and is the use justified for a particular application?" This section will examine the effectiveness of multimedia, and the following chapter will examine the decision making process for the development of multimedia-based learning applications.

A major concern when investigating the feasibility of multimedia-based applications, is being able to determine the possible advantages of the use of these systems. Multimedia applications have the potential to provide many benefits to both the student and instructor. The following is a list of some of the possible benefits: [Ref. 3]

- independent study, flexible with respect to topic
- learner controlled progress; i.e. students set their own pace
- direct involvement in various learning activities; i.e. students are able to participate in the learning experience through interaction and simulations
- systematic practice through simulations
- immediate feedback on performance
- privacy, as students use learning systems individually
- reduced cost of instruction per student
- shorter learning times (30%)
- collaboration

When examining the effectiveness of computer-based learning, one of the most important points to consider is that media or technologies itself does not make the system effective or ineffective. A study done in 1988 by an early computer-based learning advocate, James Kulik, indicated that *the effectiveness of computer-based learning was not determined by the technology or superiority of the media but by the soundness of the instructional design*. His findings showed that students learned slightly more using computer-based learing than the traditional in class instruction. In addition, the study noted that learning differences were negligible where the same instructor taught both the course and authored the computer-based learning. This is the primary reason why care must be taken when developing learning applications; one can be so overwhelmed with the technology that one neglects instruction principles. What makes effective instruction is not the medium but the instructional methods that guide the way the medium is used. [Ref. 3]

C. IMPORTANCE OF INSTRUCTIONAL METHODS

Methods are the instructional techniques that facilitate learning. Media can be described as the means for implementing those methods. If these methods are implemented improperly, it is possible to render any medium ineffective. Dr. Ruth Colvin Clark, a well respected authority in multimedia-based instruction, summed up the importance of instructional methods and instructional media by stating:

> Because of the wide variety of methods that are deliverable by means of a computer, instructional design and skills are especially critical for effective multimedia instruction. [Ref. 4]

D. THE FOUR INSTRUCIONAL ARCHITECTURES

According to Dr. Clark, multimedia instruction can be broken down into four architectures: receptive, directive, guided discovery and exploratory. It is her contention that the first and most important step is to determine the proper architecture and implement it accordingly. [Ref. 4]

1. Receptive Architecture

In this design, the instructor provides the information the user is expected to learn. The premise is the user learns new information as it is received. Briefings and linear video are examples of this type of architectures. This architecture provides very little by way of user interaction.

2. Directive Architecture

This architecture is characterized by a collection of short lessons including definitions, examples, and practice exercises. This is one of the architectures used in this study.

3. Guided Discovery Architecture

Guided discovery is described as an architecture where the learner is immersed in a job-like atmosphere and, with various support options, the user is prompted to solve various problems. The support options often include wizards, tutorials, and Reference information.

4. Exploratory Architecture

In this architecture, the user is free to access the vast selection of information. Today, it has become more common for the source of this exploratory information to be the Internet.

Table 1 shows all of the architectures and some possible applications.

Architecture	Features	Goals	Sample applications
Receptive	Provides linear information—typically with low learner control and few interactions	To inform or motivate performers	Briefings, marketing summaries, overviews
Directive	Short lessons Frequent practice Corrective feedback Simple to complex	To teach procedural skills to novices	Training on new computer systems, mathematical computations, and so forth
Guided discovery	Problem-based Situates learning in job-like environment Uses simulation to compress experience Errors are encouraged	To build expert-like problem solving knowledge and skills To accelerate expertise in principle-based domains	Acquiring skills in principle-based domains such as designing computer programs, deciding whether a loan should be made, evaluating data for specific criteria, and so forth

	Support is provided through coaching and expert models		
Exploratory	High learner control Provides rich environment for learners to explore Provides effective navigation for learner orientation	For Reference or for training of learners with good self-regulatory skills	Learning a new programming language, researching information, and so forth

Table 1. Instructional Architectures [Ref. 4]

E. IMPORTANCE OF INTERACTIVITY

Interactivity is one of major attractions of multimedia-based learning. It enables the user to be involved with the program, and gives the user a sense of control. At the same time, in order for the interactions to be effective, the designer should choose these interactions carefully. Dr. Diane Gayeski warns,

> Another key to effective interactions is the "team" approach to the development process…the design cannot be done in a vacuum. Interactive programs have got to be relevant and comfortable for the viewer - even something slightly "off", like unfamiliar terminology… can throw off the whole atmosphere of the interaction. Interactive programs require even more careful attention to detail and style than do tradition linear presentations such as video or workbooks. [Ref. 1]

She also breaks interactivity down into two fundamental elements, identification and attention.

1. Identification

Dr. Gayeski claims that interactivity allows the user to identify with the program. Her reasoning is that, by requiring the user to interact with the program (answering questions, advancing through screens, choosing topics), he/she will develop a vested interest and thus identify with the program, feeling like a contributor to the final outcome of the program. An example she cites is a class or seminar where the user is required ask and answer questions. Usually, the user feels more involved and presumably retains more information. [Ref. 1]

2. Attention

She identifies the second principle as attention. By keeping the user involved, it is less likely that for the user will lose focus. Dr. Gayeski likens this attention-holding aspect to a personal conversation or tutorial with an expert, vice listening to the same individual lecture on the same subject. [Ref. 1]

III. COST AND TIME JUSTIFICATION

One of the focuses of this study was to examine the practicality of computer-based learning modules being developed by instructors and students with minimal expertise using common off-the-shelf software and tools. Development time and cost come to the forefront when considering if computer-based learning modules are to be practical.

A. TIME

Developing multimedia graphics is extremely time consuming. This fact must be considered early in the decision making process. Some estimates of development time for one hour of interactive multimedia range anywhere from 300-600 hours. [Ref. 5] This estimate does not consider the time required if the developer is inexperienced. The most important aspect to remember is that development time will vary according to the developer's experience and the software tools available. Appendix A is a estimation guide for development time. It is generic in nature but provides an adequate estimate for certain tasks required for the development of an application.

As mentioned earlier, often multimedia development times are represented in ratios. Ravet points out that at times these ratios can be misleading and take into consideration of other factors such inexperienced developers lacking the required skills or using tools inappropriately. Table 2 gives the development time for a 12 hour computer-based learning package containing, tutorial, tests and practice. The final ratio for this project was 300 hours of development for every hour of training. It should be noted that these ratios are over all types of systems. For example, the actual development time is more a function of the depth of interactivity and the complexity of the subject matter. [Ref. 3]

	Training Hrs	**Development**	**Ratio**
Knowledge acquisition	5	500	100
Tests	2	120	60
Practice	5	2980	596
Total	**12**	**3600**	**300**

Table 2. Example of Ratios [Ref. 3]

B. COST

The issues of cost and return on investment are critical in determining the feasibility of implementing multimedia-based learning systems. Usually, cost can be broken down into two categories:

1. Production

2. Hardware

For some organizations without any resources in place, the initial costs can be anywhere from $50,000 and higher. [Ref. 6]

Gayeski examined a scenario similar to the situation here at the Naval Postgraduate School (NPS). She points out that with an organization, such as a university, cost can be reduced tremendously by using the assets already in place. By using professors and students to be content experts and the developers of multimedia productions, it is possible to attain effective, cost-efficient results without the overhead of commercial support. With the possibility of hardware also being available, i.e. development workstations and user workstations, the prospect may even become more attractive to the organization. The possibility that additional expenditures for software and training workshops for the professors would be easily justified when compared with the benefits of the course being taught via computer-based training. [Ref. 1] This was the situation here at NPS. We implemented a multimedia design workstation for under $10,000. The bulk of the money was used to purchase software, since the hardware was already in-place.

When examining cost, the benefits must be taken into account. One comparison that is appropriate to consider is the cost of the traditional presentation verses the cost of technology base training. Reynolds has developed three forms designed to address the issue of cost via traditional methods and technology-based methods. Table 3 is a development cost comparison chart for an instructor-led course with strong support from technology based learning, Appendix B contains a delivery cost comparison chart and a return on investment (ROI) checksheet that provides some fundamental analysis of anticipated costs. [Ref. 6]

Development cost comparison chart item	**Instructor-led Costs**	**TBL Costs**
Course development time • Conversion of existing course • New development		
Graphics • Animation • Drawings • Overheads		
Text production • Word processing • Reproduction		
Video production		
Totals		

Table 3. Development Cost Comparison Chart [Ref. 6]

Gayeski, gives some possible situations that justify interactive media learning: [Ref. 1]

- When the "real thing" is too dangerous, costly, or inaccessible to present.
- When personal computers are readily accessible.

- When user's responses, scores, or participation must be documented. Interactive systems can provide feedback as to which items were selected most often, and can store, request and transmit information.

C. POSSIBLE PITFALLS

With all of the possible benefits multimedia courses can deliver, there are several pitfalls the designer should attempt to avoid.

1. Sink-hole for Time and Money

The development of multimedia is time consuming by nature. In addition, if the designer does not have a detailed idea of what he/she wants to accomplish, the development can increase drastically. If the production is developed without outside input, it quite possible that some of the interactivity may be ineffective, thus causing additional development time. Software not performing as advertised, glitches and troubleshooting bugs can account for projects going over desired production time. Since multimedia is relatively new for some organizations, trying to do and buy too much at one time can account for tremendous forfeitures of funds in unnecessary equipment and labor.

2. Secondary Benefits Lost

When multimedia is used as substitute for personal interaction, some benefits are lost. In traditional courses, students have a chance to interact and gain knowledge from interacting with each other and the instructor. At times, information on the subject matter is gained above and beyond the required core curriculum. In addition, students use the skills required for daily social interaction. With technology-based learning, many of these secondary benefits are eliminated.

3. Legal/ethical Problems

Technology-based learning can pose special legal and ethical problems. Security of personal information, such as grades, is one topic that needs to be addressed. If a computer-based course is based on a text in which the author is not part of the development team, copyright laws and trademark issues need to be researched. The use of prefabricated graphics can also become an issue of copyright infringement. If testing is given in conjunction with the course, various methods of enforcing standards and ethical behavior need to be considered.

4. Orphaned Systems

With technology changing so rapidly, many different types of tools and standards are available today but gone tomorrow. The key is to be able to develop productions that will utilize the technology in ways that will allow them the maximum life cycle and to be used on the most common machines. If the developer fails to plan appropriately, it is possible for the organization to be left with obsolete equipment and unusable courses.

The next chapter, Software Selection, describes the process of selecting the software required to develop multimedia-based productions for learning applications. It examines and gives comments on the software used to produce the modules used in the EC 3840 lesson plans and simulation.

IV. SOFTWARE SELECTION

This chapter is intended to help others who may be considering different types of software. The choices and recommendations contained in this chapter may not be the best choices for all designers or applications. The first section describes the software requirements for the development of multimedia productions. A brief description of the purpose and function of the application is given. The next section describe the software used to implement the multimedia modules. Each application is given a brief description and my comments on the functionality and ease of use conclude each of the applications description.

Development software is important for the implementation and maintenance of a web-based site. Today, there are many software choices available for applications in multimedia and web design. A wrong choice can be expensive in both manpower and software costs. It is imperative that software requirements be defined at the earliest stages to insure that wise choices are made later. Once specific requirements have been defined for the software, it is necessary to decide on the specific types of applications that will be used to implement the web-site. Finally, it is necessary to determine the criteria in which the specific software applications will be chosen.

A. SOFTWARE REQUIREMENTS

In the initial stages of the design process, it is necessary to establish development software requirements. Such requirements serve as guidelines to assist in purchasing the proper software for the development of the Web-site. These requirements are general, but provide an adequate first step in the selection process.

- **Commercial off-the-shelf.** All software used should be commercially available.

- **Non-proprietary.** Ideally, all the applications used in the development of a web-site containing multimedia elements should use the standard protocols and file formats used by web-browsers for the delivery of multimedia over the web. Proprietary software packages, which use non-standard file formats, as a rule should be avoided. An exception to this requirement, would be when the designer can be sure the users have the necessary hardware and software (i.e. plug-ins) to run the application.

- **Support.** Support via online documentation, telephone and printed media is essential to the successful development of the web-site. Answers to development questions in a timely manner are crucial for efficient implementation and time management. Support also consists of the availability of other technical experts. At NPS other unique support assets are specific subject matter experts, instructors, and students.

- **Cost.** A bottom-line cost must be established to preclude unnecessary funds from being spent on software that may be overpriced or more powerful than required for the specific applications.

- **Ease-of-use.** Since our focus is on novice authors developing these web based learning sites, ease-of-use is an important requirement. Ideally, the interfaces should be intuitive, interactive and require a minimum training period.

The general requirements listed above are intended to be used as guidelines while researching software products. In order to effectively research software products, the designer must also decide on the specific types of applications required to achieve the desired goals.

B. SPECIFIC APPLICATIONS

Unfortunately, the design and implementation of the EC 3840 web-site was not accomplished with one software package. It is possible, however, to be able to implement the entire site utilizing one vendor. The use of a single vendor tends to reduce incompatibility problems common when one uses different vendors. This philosophy may not always lead to the most efficient course of action. Due to certain features unique to specific software packages, it may be more efficient to utilize several different vendors. Though many of the software packages were similar, some provided a more user-friendly environment in accomplishing certain tasks. As an example, suppose a database of questions is required to be accessed through the site. One web development package provides ease of use and site maintenance but no simple way to implement databases. A second package provides an efficient method of implementing databases but lacks the other benefits offered by the first package. In this situation, using both packages may provide the developer with the most efficient solutions.

In developing the multimedia-based Web-site the following applications were considered necessary in order to complete tasks:

- **Multimedia Development Tool.** A multimedia development package is required in order to implement various types of multimedia objects, such as animations, sounds, movies and a method in which these objects could be displayed over the Web (i.e. plug-in). This tool should be able to import all of the common graphical file protocols used in multimedia development; jpeg, gif, tif, bmp, midi, etc.

- **Graphics Design Tool.** A graphics design package should be used to manipulate and create graphic images using all of the common graphical file protocols.

- **Text Editor.** A simple word processor or text editor to generate the text files to be imported into the multimedia development package.

- **Voice Recognition Software.** This multimedia-based Web-site uses no sound because most of the workstations available to students on campus are not configured to provide sound. Therefore, many of the modules require significant amounts of text to explain certain concepts. Voice recognition software is a means for rapidly developing and editing text objects that can then be imported into the multimedia development packages.

- **HTML Editor.** The HTML editor is the backbone of the Web-site development. Therefore, it must provide several key requirements. Ideally, the editor should be able to assist the designer in Web page development as well as maintenance.

- **Computer-based Instruction Software.** Due to the popularity of computer and web-based instruction, several vendors offer specific packages that assist in implementing these sites and productions. Some common features among these packages include templates that provide a standard interface throughout the site and typical navigation options that have proven to be effective in other computer-based learning sites.

C. SOFTWARE SELECTED

Several different software packages were considered for the development of the Web-site. This section is intended to provide a brief description and critique of each software package used.

1. Multimedia Packages

a) Director 6.5, 7.0 Internet Studio ($1000)

Macromedia's Director is an authoring tool that is used for multimedia production. Director is a well-recognized tool that specializes in multi-media development for the Internet, CD-ROMs, and DVD-ROMs. Director is well equipped to generate combined media containing graphics, sound, animation, text, and video into compelling content. Many of the features needed to create sophisticated interactivity can be performed with simple drag and drop operations. Director also provides the designer more control to develop sophisticated interactivity with a

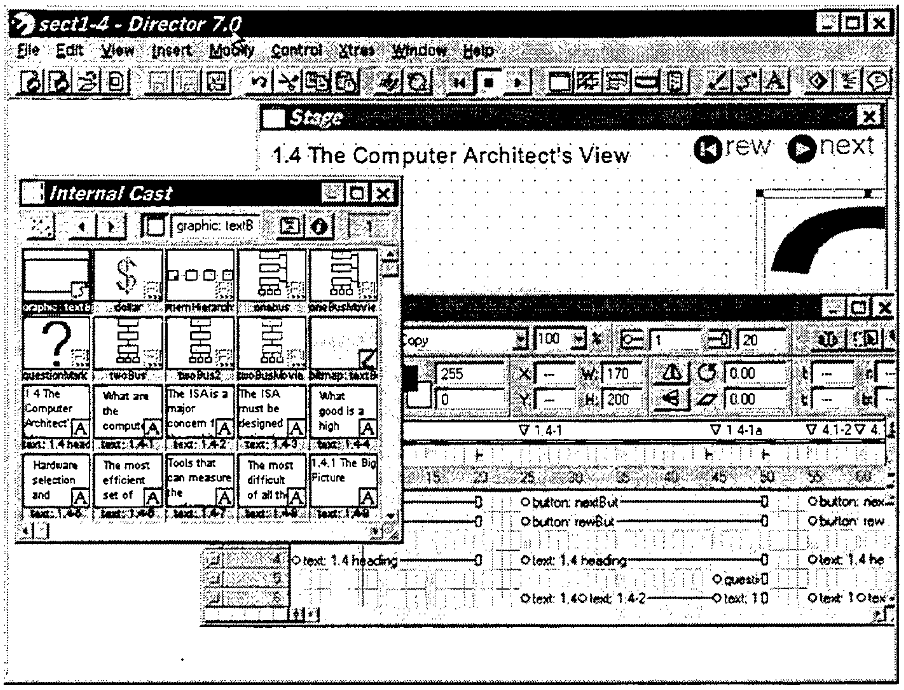

sect1-4 - Director 7.0
File Edit View Insert Modify Control Xtras Window Help
Stage
1.4 The Computer Architect's View
rew
next
Internal Cast
graphic: textB

Figure 2. Director 7

scripting language called Lingo. [Ref. 7]

Figure 2 shows Director's development environment which consists of a stage, a Score (which contains a timeline) and a cast window. The cast is a table, which consists of all the media that are be used in the presentation. The stage represents the area that the user can view the production. The Score is the interface that contains the timeline that is used to determine the animations. Basically, the cast members are distributed throughout the piece by assigning them values and positions on the timeline. This is done by dragging cast members to the stage or directly to the score (timeline). [Ref. 7]

Some of the major features of the suite:

- **Shockwave.** Shockwave is the executable file that allows browsers to view Director content. This file is often referred to as the projector. A plugin, that can be downloaded from Macromedia, is required to view Shockwave files in a browser. Shockwave is one of forerunners for multimedia on the Web and ships with Windows 95/98, Macintosh OS 8, Netscape Navigator, AOL, Internet Explorer, and Apple OS 8 CDs. [Ref. 7] What makes Shockwave ideal for the Web is its streaming and file compression capabilities.

- **Bundled Software.** The studio comes packaged with all the necessary tools to develop and distribute complex multimedia productions. Table 4 lists all the accompanying software and a brief description.

Software	**Description**
Director 7	multimedia development tool
Fireworks	web graphics development tool for bitmap and vector graphics
Soundforge XP	sound editor
3D Extreme	3D graphics development tool

Table 4. Director 7 software bundle

- **Sprite (imported graphics objects) effects.** Rotate, scale, skew, and flip sprites during playback in real time.

- **New vector shapes.** Create small, high-quality shapes with built-in vector drawing tools.

- **Animation.** With standard animation features such as onion skinning, tweening (animating objects by providing the program with the beginning and ending frame and then allowing the program to insert the additional frames), and ink effects, Director 7 provides an effective interface for creating animations.

- **Universal import.** Director 7 Shockwave Internet Studio imports over 40 multimedia file formats, including QuickTime 3, (QuickTime Virtual Reality), Flash 3, and animated GIFs. With universal import of audio, video, and graphical elements, Director has flexibility to combine a diverse set of media elements.

- **XML support**

- **Internet protocol support.**

- **Save as Java.** Allows the executeable file to be saved as a Java file, thus allowing playback on the Web without a plugin.

Overall, Director proved to be a very capable development tool. Many aspects of Director are advertised to be intuitive and supposedly, requiring no experience. *I found this comment to be overstated in most cases.* To use Director's capabilities efficiently, a good deal of time must be expended to become familiar with the environment and the other design tools. Director's native scripting language, Lingo, can add extremely intricate interactive effects. Lingo is a programming language, therefore in order to use it effectively some time must be devoted to learning the protocols and syntax.

One of the more impressive aspects of Director was the assistance received from technical support. There is a vast library of technical notes that explain many advanced techniques in step by step detail. The telephone support was outstanding and proved to be a valuable asset during the development process.

I give Director a strong recommendation for continued use in the development of multimedia productions for use on computer-based learning products. It has strong technical support and a proven track record in industry. Students also have an opportunity to become more familiar Director by enrolling in the CS 3202, Introduction to Multimedia.

b) Flash, Macromedia ($350)

Macromedia's Flash is another multimedia production package. Flash has become the standard for interactive vector graphics and animation on the Web. Web designers use Flash to create resizable, and extremely compact navigation interfaces, technical illustrations, long-form animations, and other graphics. Graphics and animation will anti-alias and scale based on the viewer's screen size, providing high-quality viewing. Movie clip and button actions create sophisticated interactivity without scripting. [Ref. 8] Figure 3 shows the interface of Flash.

Some of the features include:

- **Common drawing tools.** Most of the multimedia production tools interface consists of complex collage of menus and windows. Flash uses an interface similar to a graphics program, using familiar tools like brushes and pens.

- **Sound.** Flash supports sound and sound streaming.
- **Shape morphing.** Flash enables the designer to graphics and text. This is a feature that would require extensive Lingo programming in Director.

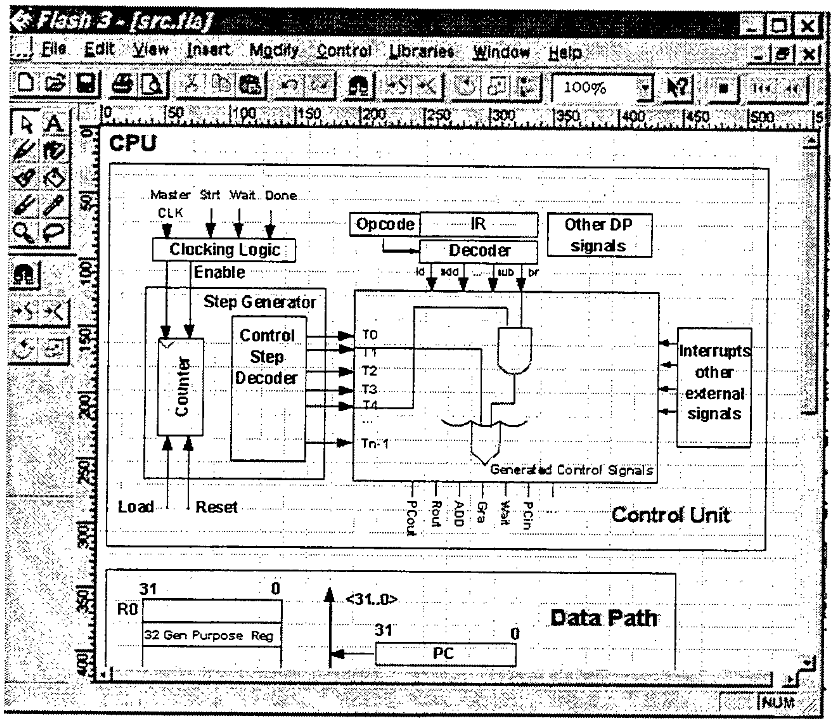


Figure 3. Flash

The major difference between Director and Flash is Flash is designed for small-scale animation productions. One major advantage of Flash is that it is easier to learn. The interface is much simpler and more intuitive. The importance of Flash with respect to the overall production is that Flash movies can be imported into Director. This enables the designer to create small modular movies to compose productions that are more complex. Flash movies retain all of their interactivity and are able to communicate directly with Director via Lingo commands. [Ref. 8]

I found flash useful. It has an efficient way to design graphics more quickly than attempting the same tasks in Director. Flash is well supported on the Web; there are user groups and books. Macromedia's Developer's Center was extremely helpful in solving many problems encountered.

2. Graphics

a) *Fireworks, Macromedia ($499)*

Macromedia's Fireworks is one of the most recommended Web graphic design software packages available. Fireworks' suite of text, design, illustration, image editing, URL, JavaScript, and animation tools has the capability to create all the graphics requirements for a production, without jumping from application to application. Designers can optimize and preview final output in most of the common graphic formats. [Ref. 9] This is extremely important because, with integrated applications like Fireworks, the designer can become comfortable and familiar with one working environment. Thus, he/she is able to increase design efficiency. Fireworks supports both bitmap and most vector formats. Figure 4 shows the interface of the program.

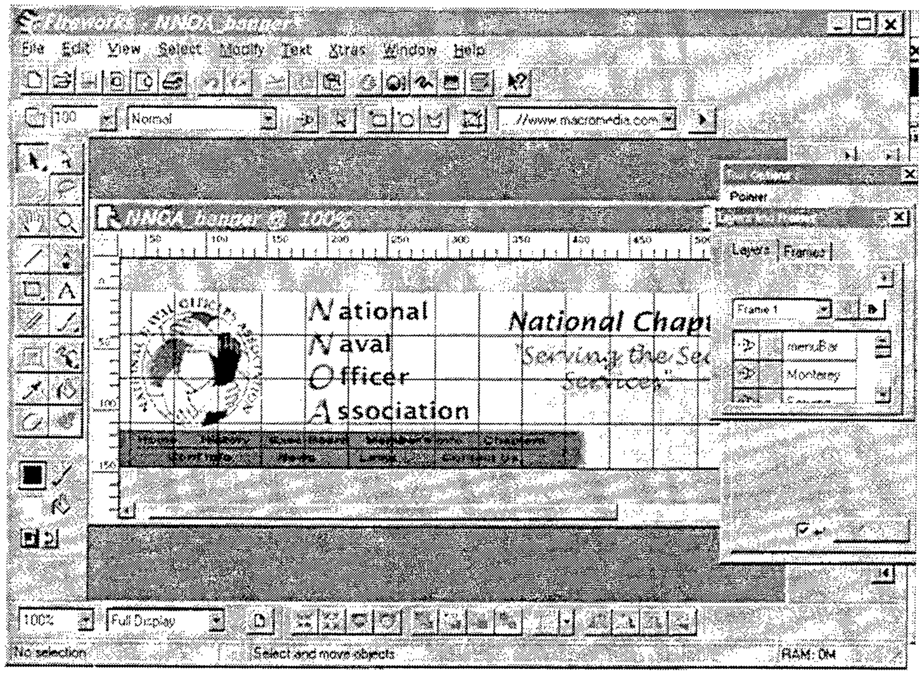


Figure 4. Fireworks

Some of the major features include:

- **Export preview.** Prior to export any graphic, the result can be previewed in the exporting format prior to saving the file. This way the designer can choose the optimal format for a given application prior use.

- **Automated image mapping and slicing.** This feature allows complicated graphics to be sliced and broken down into different pieces and file types.

- **Supports Java rollovers and tweening.** Fireworks can be used to create animated GIFs and Java rollover without the user writing any code.

With the benefit of having used several other graphics/illustration applications, I found Fireworks reasonably easy to use. Many of the interfaces are similar to the lower end graphic programs. This is convenient for the novice user because he/she will be able to be productive at the beginning.

3. Voice Recognition Software

a) Naturally Speaking Preferred, Dragon Systems Inc. ($150)

In an application where enormous amounts of text will be generated, I recommend using voice recognition software. This was important to me in this design because once the decision to base the tutorials on text vice speech was made, it was necessary to generate a large amount of text. The software reduced the time required to input and edit text for the production. Naturally Speaking performed well. The program allows the user to read text into just about any text generation application.

Accuracy was about 85-90 percent. This was dependent on the surroundings and how much time the user spends training the software. There are several lengthy selections to read into the computer to start the initial training of the software. Initially, the process was awkward, but after some additional training and practice, it proved to be a worthwhile investment.

4. HTML Editor

a) Dreamweaver, Macromedia ($300)

Macromedia's Dreamweaver is a visual editor for creating and managing web pages. One of Dreamweaver's strengths comes from being able to generate pure HTML code that is cross-platform and cross-browser compatible. Another advantage of Dreamweaver, is that it imports HTML documents without reformatting the code. [Ref. 10]

Dreamweaver is intended for the intermediate user rather than the novice. Unlike FrontPage, Dreamweaver contains no pre-designed templates for quick development of web pages. But unlike FrontPage, the designer can be reasonably certain that the documents created will function on all current browsers. In addition, Dreamweaver generates HTML that is easier to understand and is in a more logical format.

Some of its features include:

- **Design in layers.** Layers gives the designer a "what you see is what you get" (WYSIWYG) environment. Unlike designing with frames, designing in layers allows graphics and objects to overlap. This allows for exact placement of objects and a predictable outcome when viewed in a browser.

- **Animation.** Dreamweaver allows the designer to deliver basic animations without any programming. Dreamweaver uses dynamic HMTL and Java to accomplish the animations.

- **Template design.** Allows for the design of templates to provide the site with a uniform appearance.

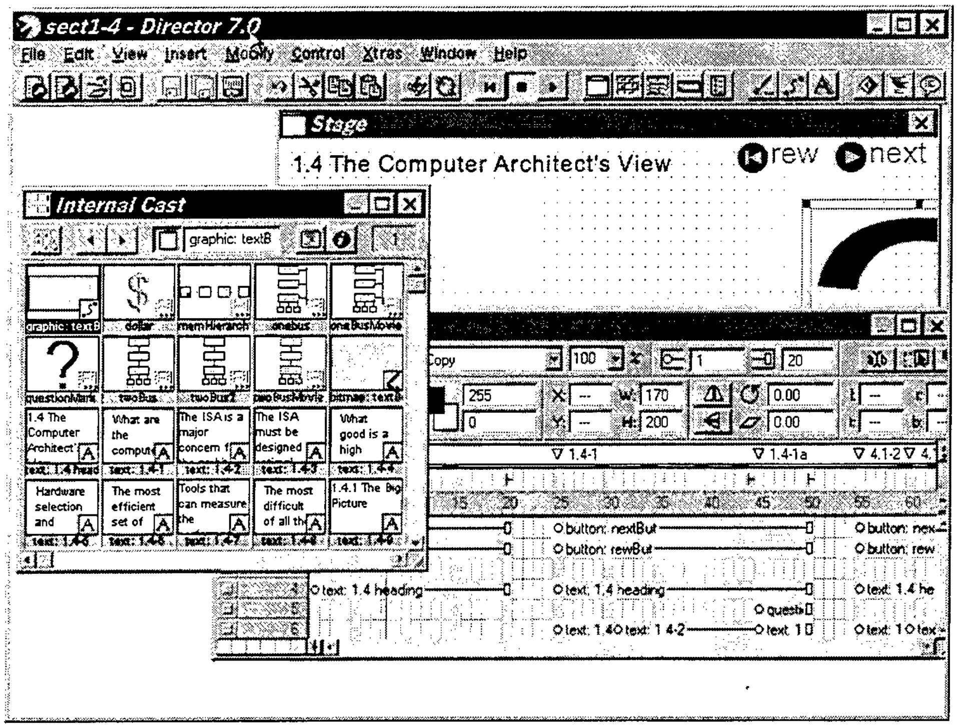

sect1-4 - Director 7.0
File Edit View Insert Modify Control Xtras Window Help
Stage
1.4 The Computer Architect's View
rew
next
Internal Cast
graphic: textB

Figure 5. Dreamweaver

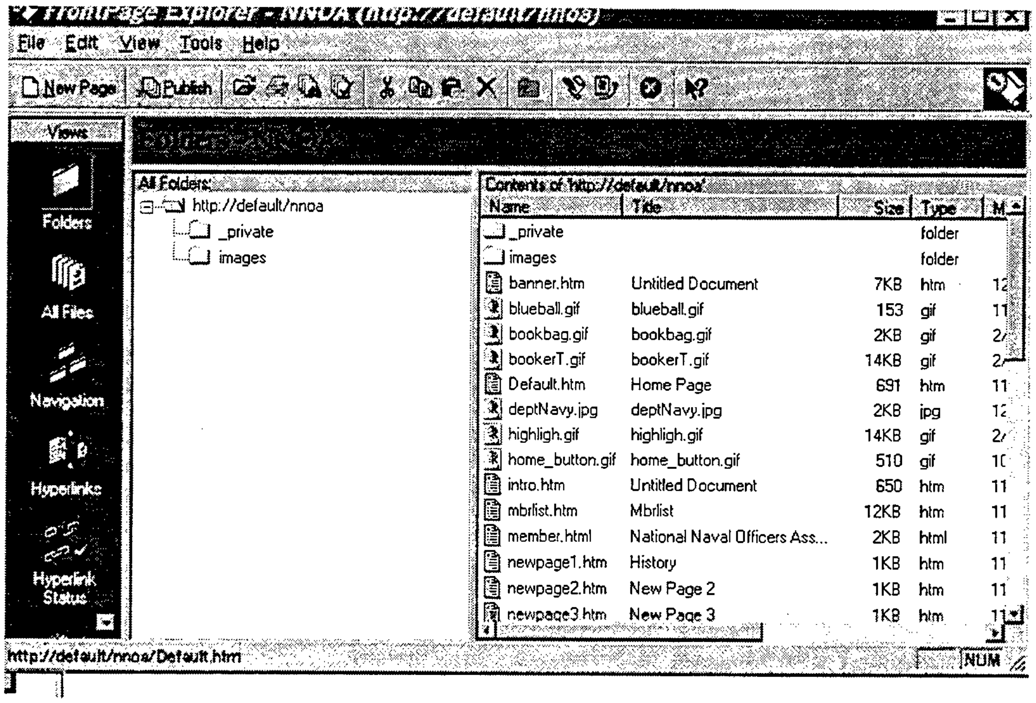
I would recommend Dreamweaver for someone who has time to learn or who is already familiar with the basics of HTML design. *Conversely, if an entire site must be developed quickly and the designer has no experience, I would recommend FrontPage.*

b) FrontPage 98, Microsoft ($149)

FrontPage is a web-authoring tool and site manager designed to rapidly create a single page or an entire site. FrontPage actually consists of FrontPage Explorer, the site management tool, and the FrontPage Editor, the HTML editor. FrontPage is an impressive tool that can assist in developing web-sites for the novice as well as the professional designer. [Ref. 11]

Some of FrontPage's features:

- **Themes.** FrontPage introduces themes, which provide a common look and interface for an entire site.
- **Wizards and templates.**
- **Extensive ActiveX and Java support.**



Name	Title	Size	Type	M
_private			folder	
images			folder	
banner.htm	Untitled Document	7KB	htm	12
blueball.gif	blueball.gif	153	gif	11
bookbag.gif	bookbag.gif	2KB	gif	2/
bookerT.gif	bookerT.gif	14KB	gif	2/
Default.htm	Home Page	691	htm	11
deptNavy.jpg	deptNavy.jpg	2KB	jpg	12
highligh.gif	highligh.gif	14KB	gif	2/
home_button.gif	home_button.gif	510	gif	10
intro.htm	Untitled Document	650	htm	11
mbrlist.htm	Mbrlist	12KB	htm	11
member.html	National Naval Officers Ass...	2KB	html	11
newpage1.htm	History	1KB	htm	11
newpage2.htm	New Page 2	1KB	htm	11
newpage3.htm	New Page 3	1KB	htm	11

http://default/nnoa/Default.htm

Figure 6. FrontPage

FrontPage was the most intuitive web page design software I used. It was fast and simple to develop web pages. However, intricate tasks were difficult to implement. Also, it was difficult to interpret what the HTML code was doing. This made it difficult to make minor changes directly to the HTML code. In addition, much of the code generated would only function on Microsoft servers or on their browser, Internet Explorer. *Despite those flaws, I would recommend FrontPage to any user who did not want to do any HTML coding and needed to develop a site in a minimal amount of time.*

5. Computer-based Instruction Packages

a) Authorware 5 Attain, Macromedia ($2400)

Authorware is a powerful tool for developing intricate types of interactive multimedia. Authorware has been a leader in the computer-based training (CBT) authoring industry for many years. One of the main advantages of Authorware is its ability to use a variety of media in the production process. The package provides a complex interactive and navigation functions. Authorware adds some additional features from previous versions which makes it a viable solution for web based training. [Ref. 12]

Unlike Macromedia Director, Authorware is optimized for instructional applications. It offers eleven user input/interaction types, making it a flexible environment for technology based traning and testing. Text handling is vastly superior, including on-the-fly text formatting, hypertext features, and user-defined hot typefaces and styles.

Recently, Macromedia acquired Pathware (formerly Solis Pathway), a large-scale computer-managed instruction (CMI) system that manages access to courses and tracks student progress. The latest versions of these tools, Authorware 5 and Pathware 4, introduce a variety of new features which make computer-based training possible for smaller institutions.

Every element of an Authorware project, from graphics and audio files to decision-making and motion, is represented by an icon. The authoring process consists of dragging and dropping icons into place in a flow chart on the main document window. The designer can step through playback one icon at a time. This is a useful tool for troubleshooting productions.

Designers with no programming or scripting experience will appreciate the simplicity of this approach. Experienced programmers and scriptwriters, on the other hand, might find it time-consuming and cumbersome. Authorware strikes a balance between these two camps, augmenting the icon authoring with a scripting environment. Designers familiar with Director will find a totally different interface in Authorware. Being that Director is a timeline based design application and Authorware is icon based design tool, the transition between the two products can be awkward and frustrating.

Authorware is an extremely powerful tool that can assist in the laying the groundwork for computer-based learning application, but it does not come without trade-offs. The learning curve is steep, despite excellent online help, Show Me movies, and QuickStart templates. Being that Authorware is an icon-based application, large productions can become difficult to manage due to the number of icons generated. Also, some tasks that could have been simplified into a flexible dialog box require scripting instead. In any case, it's unlikely that a complete project can be authored without resorting to scripting at some point. [Ref. 12]

The latest version of Authorware has added a feature called Knowledge Objects. Knowledge Objects are intended to be self-contained templates for some of the more complicated tasks that would require intensive scripting. The intent is to give the first-time user a starting point. These objects range from designing the general navigation of a structured course to the development of various types of interactive questions, i.e. true and false, multiple choice, short answer and implement specialized functions.

An important point that needs to be addressed is, Authorware's primary use is not for internet applications. Its true strength comes from generating executable programs that are usually distributed on a disk or CD. One aspect that should be remembered when using Authorware to design

Knowledge Objects may not be available. An example of this was that the Knowledge Object used to build the interface had an option to enable security settings for the users. This feature was not available for use on the web. This does not imply that security features are unavailable for productions distributed on the web; it means that additional scripting would be required to implement the feature. Another danger in using the knowledge objects or templates without understanding the principles behind them is the unpredictable outcome and the inability to implement minor variations from the template.

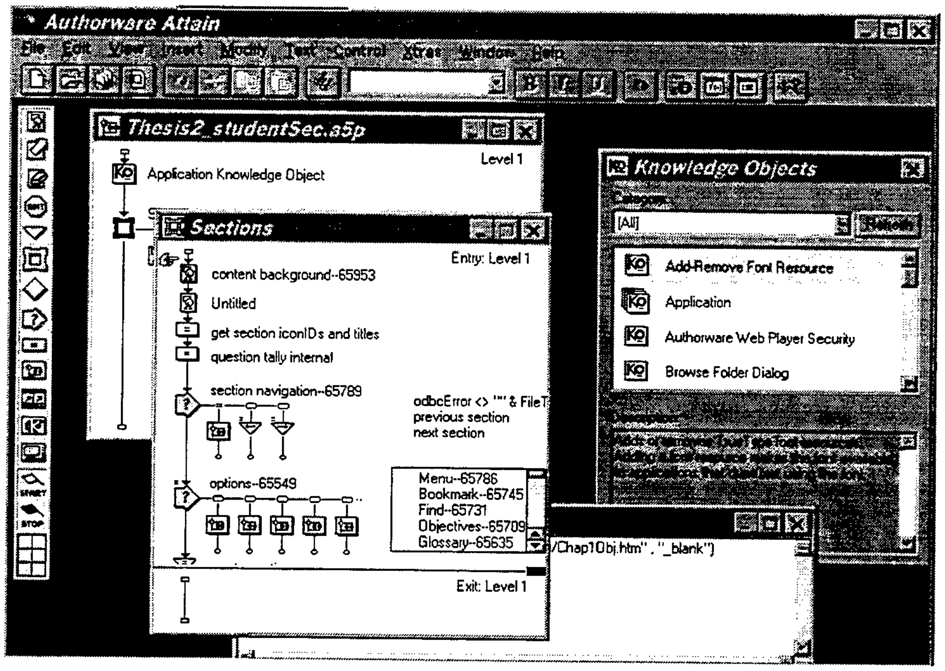


Figure 7. Authorware

D. COMMENTS

Overall I was satisfied with the software used to develop this project. Several of the packages were newly released or revised. This led to several problems when importing new or revised formats into some applications. Many of the programs had been released for only a few months. Some issues involving formats and compatibility were unknown and undocumented. This had a major effect on production time. Several issues arose regarding compatibility problems between the multimedia packages importing media. Most of these were due to changes in vendor's standard or the release of an upgraded product. An example of this was when I attempted to import Director 7 movies into Authorware. Director 7 movies were not supported by Authorware. The developers were in the process of creating a fix for the problem. They provided me with a beta version of a plugin that was supposed to allow for the import of Director 7 movies. Even after installing the plugin I was unable to import Director 7 movies into Authorware. This was a major issue in the development of the modules. As mentioned earlier, one of reasons for choosing Authorware was the ability to directly import media from Director.

Table 5 lists all the software, function and the recommended use for the software used in the study.

Software	**Function**	**Recommended Use**
Director 7	Multimedia suite	ideal for complex interactions and animations
Fash 3	Multimedia suite	ideal for simple animations and basic interactions
Fireworks	Grapics development	ideal for creating and modifying grapic files
Dreamweaver	HTML editor	ideal for intermediate skilled designer to develop web-sites
FrontPage	HTML editor	ideal for novice designer to develop web-sites
Authorware	Computer-based instruction development suite	ideal for designing and computer-based learning applications

Table 5. Software used in study

Chapter 5, The Development Process, introduces a development system that has been developed for the design of multimedia-based instructional productions. The principles used in this system are key to the successful delivery of instructional media.

V. THE DEVELOPMENT PROCESS

This chapter is intended to introduce the process that the vast majority of instructional media incorporate, Instruction Systems Development (ISD). After describing the ISD process, the actual design process used in the thesis will be described.

A. INSTRUCTION SYSTEM DEVELOPMENT (ISD)

ISD can be viewed as the technology of instruction. ISD is a term for a variety of processes used for planning and developing instructional programs. These processes are designed to ensuring that the user is taught the skills and objectives essential for the successful completion of a group of tasks or other applicable course requirements in a cost-effective manner. Gayeski claims that if these processes are followed, the instructional designer can be confident that the educational package will be effective. [Ref. 1]

ISD is based on a "systems approach" toward the development and design of instructional media. The ISD model uses distinct phases to modularize the development process. The phases are as follows:

- Analysis
- Design
- Development
- Implementation
- Evaluation

ISD requires extensive documentation at each of the design levels. This is essential when developing courseware in teams or when someone must revise courseware other than the original author. It allows for smoother revisions. A brief description of the phases of ISD follows:

1. Analysis

The analysis phase typically is used to define the needs and constraints of the project. In identifying the constraints, various restrictions can be identified which could impact possible solutions. The analysis phase is also used to determine the resources available, both technical and financial. The target audience must be determined and conclusions drawn on the audiences' previous experience, skills, and required expertise. This phase also accesses the job and makes recommendations on the functionality of project. It is possible in this phase to determine that the project is not feasible due to constraintsb on time, resources or technical expertise. Another possibility could be to scale the project down from an entire course to individual tutorials on certain problem areas. After an analysis of needs has been accomplished, the next step is to develop specific objectives to address the problem areas. These objectives are important because they provide the designer with a means of evaluating the effectiveness of the course. This phase is usually documented with a formal or informal Analysis Report.

2. Design

The design phase usually includes determining the specifications of the learning activities. Decisions regarding the media best suited for the project are addressed in this phase. These learning activities may consist of the full spectrum of media and methods to be implemented. Typically, a search is conducted for existing media and instructional materials. This is important because the more existing media and instructional materials able to be reclaimed, the more cost effective the project. These media can range from slide presentations and video taped lectures to grahics created for other projects. The design phase is documented with the Design Document. [Ref. 1]

3. Development

The development phase is the most unique. The reason for this is the development phase is dependent on the many of the choices made in the design phase. Choices regarding the media types and the learning methods are responsible for the components of the development phase.

The development phase is probably the most intensive of all the phases. Storyboards, scripts, graphics must be compiled and put into digital form. This is phase where the subject matter experts must be consulted to ensure the content is correct and being presented in the best format for mastery of the information.

During the development phase, the designer should keep in mind some basic strategies that are commonly used in multimedia design. Gayeski lists six points that she feels the designer should keep in mind while developing interactive course work: [Ref. 1]

- Think of your interactive design task as creating an instructor rather than instruction.
- Create conversations. Not a lesson or lecture.
- Ask questions first before diving into providing detailed information.
- Construct test questions so that most users get them wrong. An appropriate level of challenge must be provided.
- Do not strive to develop just one correct or best presentation of the content; try to accommodate different kinds of presentations and different kinds of learners.
- Develop the question first, then develop the content around the questions.

Ideally, formative evaluation should occur during the development phase. This is done by individuals or small groups using course materials and then providing feedback to the developers and the subject matter experts for possible problems or suggestions to improve content. The actual production of the final materials cannot be completed until the formative evaluation is complete and all problems have been rectified. The final product of this phase is the completed courseware. [Ref. 6]

Reynolds suggests that designers should address these four major questions during the development phase: [Ref. 6]

- What resources (personnel, time, etc.) are used in developing a product?
- How can the product be improved?
- What is the immediate effectiveness of the product?
- What is the impact of the product?

4. Implementation

Implementation is the actual delivery of the final product to the intended audience. This could be burning a CD, posting lessons on the web, and support for the users. Support should encompass developing a help system as well as establishing a maintenance system.

5. Evaluation

Evaluation consists of two parts, formative and summative. The summative evaluation is used to measure the effectiveness of the course in solving the problem that was identified in the analysis phase. This phase is accomplished while course is in use. Problems and improvements identified in this phase are corrected and implemented in revisions. A more complete discussion of evaluation will be addressed in the next chapter.

B. COMPONENTS OF ANALYSIS

As discussed earlier, analysis is the first phase of the ISD processes. The analysis phase is considered by many to be the most important. A thorough analysis can prevent misconceptions regarding the objectives and processes anticipated to complete the project. It can serve as a contract to the development team as well the technical experts the courseware supports.

Figure 8 shows the table of contents for a typical Analysis Report. There is no standard format for this report and the style varies. Despite the variations in format the content is usually very similar in most analysis reports.

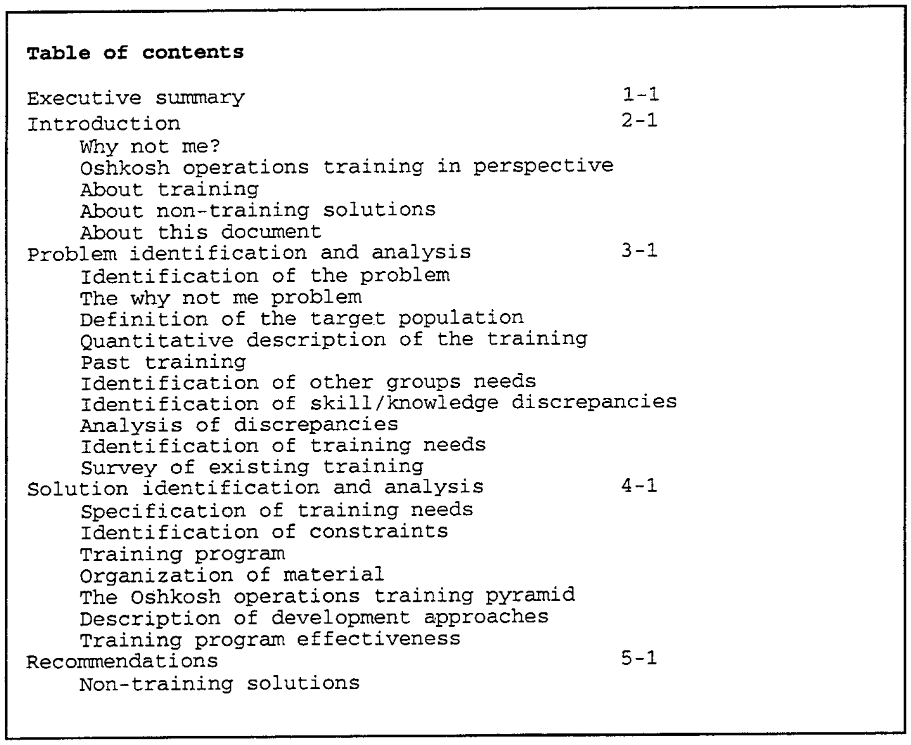


Table of contents

Executive summary	1-1
Introduction	2-1
Why not me?	
Oshkosh operations training in perspective	
About training	
About non-training solutions	
About this document	
Problem identification and analysis	3-1
Identification of the problem	
The why not me problem	
Definition of the target population	
Quantitative description of the training	
Past training	
Identification of other groups needs	
Identification of skill/knowledge discrepancies	
Analysis of discrepancies	
Identification of training needs	
Survey of existing training	
Solution identification and analysis	4-1
Specification of training needs	
Identification of constraints	
Training program	
Organization of material	
The Oshkosh operations training pyramid	
Description of development approaches	
Training program effectiveness	
Recommendations	5-1
Non-training solutions	

Figure 8. Analysis Report Table of Contents [Ref. 6]

Reynolds explains the importance of focusing on the learner. He suggests the identification of the learners prerequisite knowledge and the assets available to support the training. [Ref. 6] The analysis should clearly identify the training problem to be solved and offer a well-defined solution. In addition, a method in measuring the effectiveness of the finished product should be included in the report.

Skill/task analysis is one of the most critical key elements in the analysis report. Task analysis is "the process of arriving at a step-by-step description of all the performance elements". [Ref.6] This is necessary to allow the learner to successfully complete the lesson. This is accomplished through close coordination with subject matter experts and the development of precise learning objectives.

Reynolds offers another technique for the collection of data for the analysis report Referred to as DACUM (developing a curriculum). [Ref. 6] This technique brings all the involved parties, i.e. technical matter experts, graphic designers, evaluators…, to address all the analysis issues and resolve conflicts immediately.

C. COMPONENTS OF THE DESIGN DOCUMENT

The design document is the key to a successful development process. It is nearly impossible to design and implement a successful and revisable multimedia learning environment without an effective design document. One of the challenges of developing multimedia-based course work is to turn creative ideas into a format that will be appealing to the learner, technically correct , and revisable.

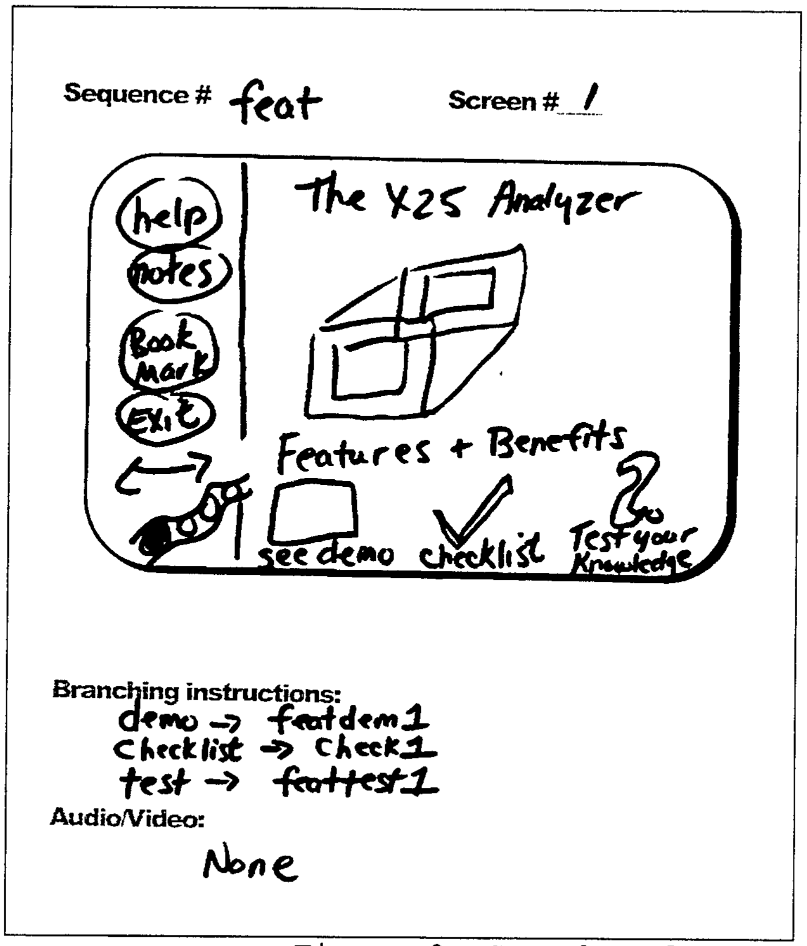
Reynolds describes several elements that the design document should contain: [Ref. 6]

- Demonstrate an understanding of the project and organization. This is accomplished by outlining a brief background of the project and organization.
- Include an explicit verbal description of each component of the course. In addition, flowcharts of the project are also recommended.
- Provide as much documentation explaining the logic behind the design and decision making.
- If the project requires more than one designer, set out the responsibilities of each individual and desired milestones.

D. PROGRAMMER READY MATERIALS (PRM)

PRM are commonly referred to as scripts and storyboards. These documents are used to ensure the content and learning methods are compatible with the technical experts or sponsors expectations. By developing these documents, the chance of time being wasted on developing inappropriate media is greatly reduced. The goal is to be able to give the PRM to any programmer as a complete specification. [Ref. 6]

PRMs can be described as forms that explain the content of the multimedia product frame-by-frame or scene-by-scene. Figure 10 shows an example of a storyboard form.



Sequence # feat Screen # 1

Branching instructions:
demo -> featdem1
Checklist -> Check1
test -> feattest1

Audio/Video:
None

Figure 9. Storyboard

These storyboards describe all the activities for the current frame. This includes possible user interactions, animations, and narrative. The images can be sketched on the form to give a general representation of what is supposed to happen during that particular frame.

The more complete the storyboards the easier the process of development. It also allows for interactions and concepts to be discussed prior to beginning any programming.

It also adds to the documentation if, in the future, a different developer wanted to revise the project.

Now that the development process has been discussed, the next chapter describes the principles of evaluation. The evaluation phase is crucial in order to be able to determine the effectiveness of computer-based learning productions.

VI. EVALUATION

The evaluation process is used to do improvement, justify the use of training to solve performance problems, and to assist in deciding whether to continue with a certain type of training strategy. One wants to be able to identify strengths and weaknesses in the training process. One of the most important reasons for evaluation is to determine if the overall objectives have been met.

As mentioned earlier, evaluation is a two part process, the formative and summative. The formative evaluation is accomplished during the development of production. The summative evaluation is designed to measure the effectiveness of the program to solve the problem that was identified in the analysis phase of the design. [Ref. 6]

A. FORMATIVE EVALUATION

Formative evaluation can be thought of as developmental testing. This type of testing is done throughout the entire developmental phase of the design. The purpose for this incremental testing is to ensure the design is valid, relevant, and appealing to the user.

Reynolds recommends that, during the formative evaluation process, the designer answer the following questions: [Ref. 6]

- Do learners understand what their options are at any given moment?
- Does the lesson maintain the learner's attention?
- Do learners accomplish the objectives of the lesson?
- Is it feasible to implement the lesson as designed?

Answering these questions early in the design process will result in a more efficient development phase. Formative evaluation can be broken down into three general stages: [Ref. 6]

1. One-on-one Trials

This is accomplished by the developers evaluating the program on a one-to-one basis. This is done when the project is near completion. The purpose of this stage is for the author/developer to determine how well the materials are likely to work. The author/developer goes through the production with the learner explaining the desired effect and then requesting feedback. With feedback from the learner, developers can assess how well the materials are achieving the desired goals.

2. Small Group Trials

This evaluation process takes on the same aspect as step 1 but with a small group. This allows the developer to assess the training materials among a more diverse group of learners. Small group trials are usually accomplished when the production is close to completion. Unlike the stage one trials, this time the developers allow the users to test the materials without interruption or explanation. Then, from feedback from users, the developer should implement modifications and improvements of the production.

3. Field Trial

Field trials are accomplished on materials that are expected to be totally free of errors and are expected to meet the objectives set forth in the analysis phase. These trial groups are larger than those discussed in phase 2. At this stage the purpose is to collect feedback to fine-tune the project.

B. SUMMATIVE EVALUATION

Summative evaluation is the collection of data to determine the effectiveness of program. This collection process is done once the course is implemented and deployed to the users. It is during this phase that the accomplishments of objectives and rate of return are assessed. Kirkpatrick describes four levels of summative review: [Ref. 13]

1. Level 1: Reaction

At this level, the evaluator asks the question, "How much did the learners benefit from the program?" This is accomplished by questionnaires addressing the learner's experience with the program. This allows the developer to evaluate how well the training environment was managed. [Ref. 13]

2. Level 2: Learning

At this level, the evaluator asks the question, "What principles, facts and techniques were learned?" This is done usually by testing whether (verbal or written). From the results the evaluator can determine whether the program allowed the learner to master the desired techniques. [Ref. 13]

3. Level 3: Behavior

The evaluator asks, "What changes in behavior resulted form the program?" An example of this would be if a tutorial was taken early in the quarter, at the end of the quarter did the learner display the desired level of skills acquired? Usually, this level would be accomplished by the surveys of individuals who evaluate the learners. [Ref. 13]

4. Level 4: Results

The evaluator asks, "What were the tangible results of the program in terms of reduced cost, increased productivity, etc.?" At this level, the evaluator must carefully scrutinize the overall effectiveness of the program. [Ref. 13]

To this point the study has concentrated on the mechanics and principles of computer-based learning. In the next chapter the development of the productions used in EC 3840 tutorial and simulation are described.

VII. EC 3840 WEB-BASED TUTORIAL AND LESSON PLAN DEVELOPMENT

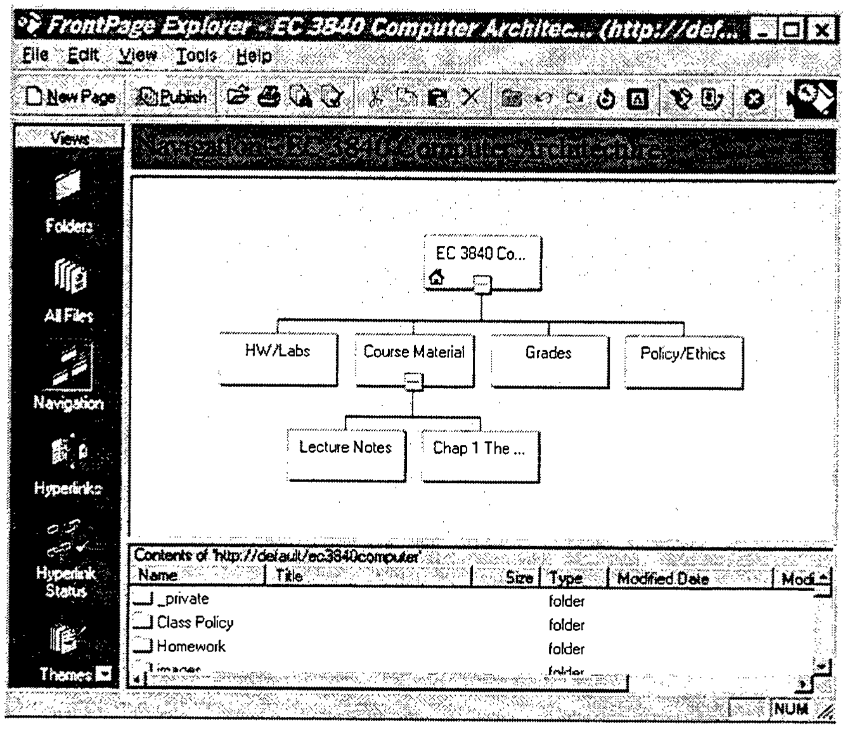
This chapter summarizes the development of tutorials and a web-site. Unfortunately, the ISD process was not followed completely. This is one reason I believe the development of the project was more challenging than expected and was more difficult to revise.

The development was broken down into five different stages. First, the HTML based site was constructed. This consisted of the EC3840 home page and supporting HTML pages. The second stage consisted of developing the framework and structure of the lesson plans using Macromedia's Authorware. The third stage was the design of tutorials using Macromedia's Director and Flash. The fourth stage consisted of importing the media from Director and Flash into Authorware. Finally, the fifth stage consisted of distributing all the media to the web.

A. WEB-SITE DESIGN

The HTML pages were designed with Microsoft's FrontPage and Macromedia's Dreamweaver. The most straightforward design implementation resulted from FrontPage. With the use of FrontPage's themes, I was able to establish a consistent "feel" throughout the entire site. Buttons, background, and banners throughout were standardized. FrontPage consists of two interlinked applications, FrontPage Explorer and

FrontPage Editor. FrontPage Explorer is used to manage site files, links, and give the designer a graphical representation of the site via a block diagram. Figure 10 shows the block diagram of the EC3840 web-site.



Contents of 'http://default/ec3840computer'

Name	Title	Size	Type	Modified Date	Modi
_private			folder		
Class Policy			folder		
Homework			folder		

Figure 10. FrontPage - site layout

1. Initial Phase

As mentioned previously, the design of the web-site was relatively straightforward. Even though the process may appear to be easy, the designers of web-sites should be sure sound design principles are being used.

Prior to starting the development of a web-site, Dr. Patrich Lynch, a recognized expert in web-development, recommends several questions be addressed: [Ref. 14]

- **What is the mission of the organization?** In the case of the site to be developed for EC3840, the purpose of the site was to provide the students and the professor another means of sharing information. This includes assignments, test scores, and tutorials. ect.
- **How will creating a web-site support the purpose?** The web-site will provide a means to communicate the desired information to students.
- **What are the immediate goals for the site?** Initially, the desire is to establish a sound framework that can be developed into a more complex site as needs are determined. Also, the site can be used as a test-bed for future development.
- **What are the longterm goals of the site?** When fully developed, the site would provide students with all the necessary information to support the course. This includes notes, interactive tutorials, grades, practice exams and other pertainent information.
- **How will success be measured?** Success will be measure by the feedback from the students via evaluations and critiques given at the conclusion of each quarter, specifically addressing the usefulness of the site.

2. Site Development Process

The development process used in the web-site design is very similar to the principles discussed earlier with ISD. The development of a complex web-site usually takes on the following five steps: [Ref. 14]

- Site definition and planning

- Information architecture
- Site design

SITE PRODUCTION CHECK LIST

Production

- What are the purpose and the goals of the site?
- Who is the target audience for the site?
- Will the site production team be composed of in-house or outside contractors?
- Who will manage the process?
- Who are your primary content experts?
- Who will the liaison to any outside contractors?
- Who will function as the long-term webmaster?

Technology

- Browsers and operating systems should your site support?
- What is the network bandwidth of the average visitor?
- Are advanced technologies required?
  - DHTML
  - Javascript
- How will readers reach support personnel?
  - Email messages?
  - Chat rooms?
- Database support?
  - User log-ins required?
  - Questionnaires required?
  - Search and retrieval from databases needed?
- Audiovisual content?
  - Video or audio productions?

Web server support

- In-house web server or ISP?
  - disk space or site traffic limitations?
  - CGI, programming, and database middleware support?

Budgeting

- Salaries
- Hardware and software
- Staff training in web use, database, and web design.
- Outsourcing fees
- Ongoing personnel support for site
- New content development and updating

Figure 11. Site Layout

- Site construction
- Tracking, evaluation, and maintenance

a) *Site definition and planning*

This is the initial stage of design. At this point the purpose and objectives of the site should all be defined. All questions asked in the initial phase should be answered. The decision on whether to design and implement using in-house resources or contract the development to an outside source should also be addressed. Figure 11 is a site production checklist that can be used as guide in site definition phase.

During the site definition and planning phase of the EC3840 site, several decision were made that would effect the choices of software, hardware and media presentation.

In defining the target audience, the user was assumed to be Internet savvy and have a basic understanding of site navigation. It was also assumed that the majority of users would access the site from workstations at NPS. With that in mind several assumptions were made concerning the user's hardware and software configuration:

- The dominant operating system would be Windows.
- The user would have access to both of the popular browsers, Netscape Navigator and Internet Explorer.
- Most systems would not have sound cards installed. Therefore the design would minimize the use of sound as a media.
- Most systems would have processors that perform at speeds above 90 MHz.

- All of the required plugins would be installed on the ECE workstations.

In general, when designing a web-site, the site should not be optimized for any particular software or hardware. The design should be targeted toward the low end or less capable system, not depending on additional hardware or plugins to be able to use the site. This point of view allows for the site to perform well on the majority of systems. In the case of the EC3840 web-site, a choice had to made regarding hardware and software configurations in order to be able to use many of the features in the software packages. Several examples follow:

- Macromedia's Flash, Director, and Authorware, all require browser plugins to be viewed by the users. These plugins require browsers versions 4.0 and above.

- Microsoft FrontPage uses some proprietory instructions that only function with Internet Explorer. Other features can only be easily implemented if the site resides on a Mircosoft server with the proper extentions.

- Both of the HTML editors and the graphic programs utilize dynamic HTML and Javascript, which are used to make web pages more interactive. Another reason for the assumption of browsers versions greater than 4.0.

- For superior performance of the animations generated, Pentium class processors with MMX are recommended.

With the assumption that most of the users would access the site from the NPS workstations, bandwidth limitations were not a major concern. With respect to the HTML code, FrontPage makes no attempt at optimizing the HTML code prior to distribution. Macromedia's products can be designed with streaming options that allow the media to start prior to the entire file being downloaded. These streaming options can still significantly increase download times if accessed through slower connections.

b) Information Architecture

The term "information architecture" refers to the content and organization of the site. For EC3840, the information archecture is shown in figure 11. A hiearchical organization was implemented. This can be seen by moving through the most general information in the site, the EC3840 Home page, to the more detailed information in the lower levels of the site. [Ref. 14]

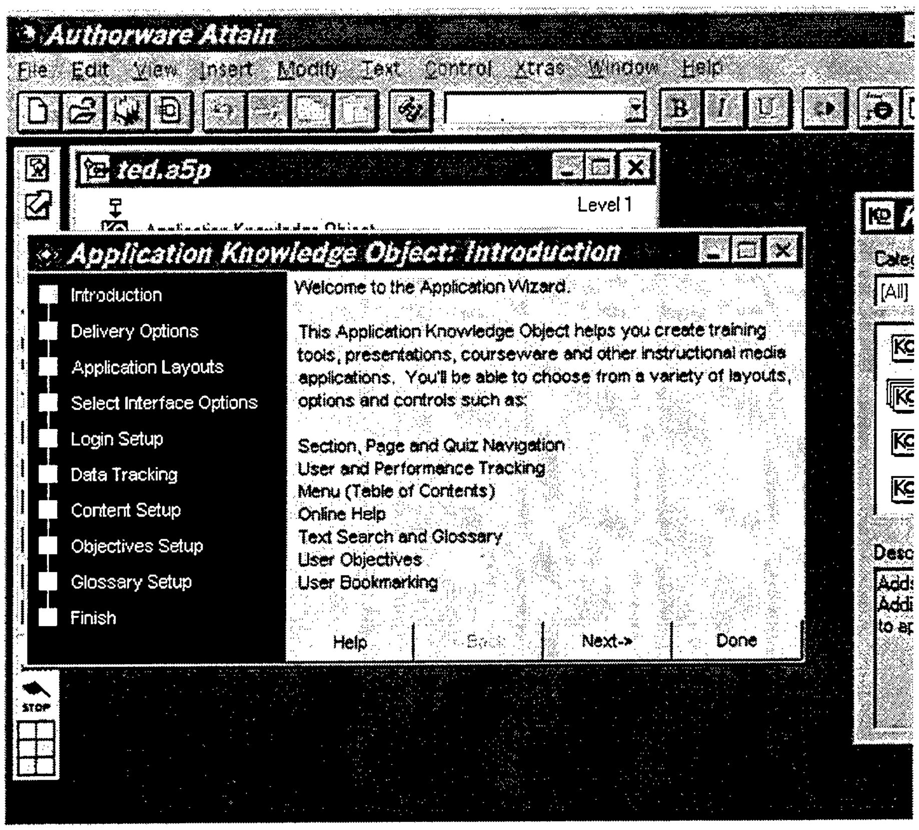
c) Site Design

This is where the site's look and feel are developed. By choosing one of the many themes available in FrontPage, a professional interface was developed without problem. One consequence of using FrontPage's themes is the addition of a significant amount of additional files and proprietory code. For the most part, the code is uneditable by the designer. FrontPage adds many files and folders to the directory structure. This makes troubleshooting and debugging almost impossible if there is a problem in the interface. In the next section the emphasis is shifted from the web-site to the development of the multimedia productions.

B. TUTORIAL DEVELOPMENT/DESIGN

After the decision was made to develop a tutorial of the chapter 1 of the EC3840 text was made, the next step was to decide on which tools would be used for development. As mentioned earlier, Authorware's strength is as a development tool for use with training. It provides complex interactions and many different navigation techniques. The software provides student monitoring, progress reports, statistics about time spent on particular topics and other features that make it an attractive option for the interface of the project. Authorware has the capabilities to produce animations, drawings, graphics, and other multimedia-based productions. But implementing these features can be awkward and usually can be done more efficiently with a software package specialized for the task, such as Director or Flash. Since Authorware is able to import most of the popular media formats from a variety of other media, it is ideal for presentation of the tutorial productions.

1. Interface



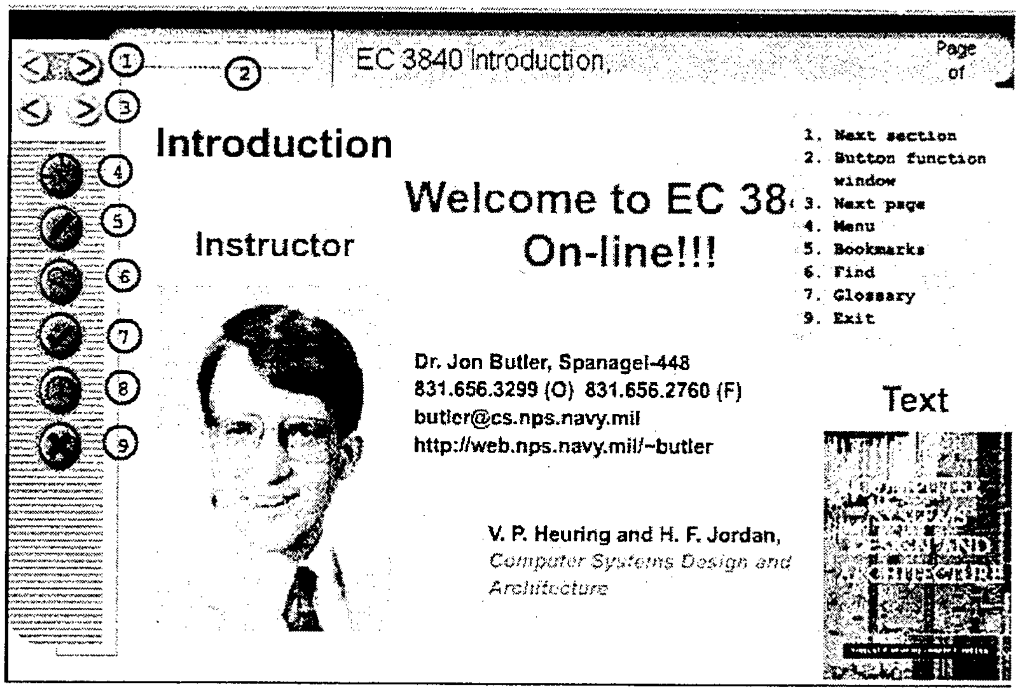
Welcome to the Application Wizard.

This Application Knowledge Object helps you create training tools, presentations, courseware and other instructional media applications. You'll be able to choose from a variety of layouts, options and controls such as:

Section, Page and Quiz Navigation
User and Performance Tracking
Menu (Table of Contents)
Online Help
Text Search and Glossary
User Objectives
User Bookmarking

Figure 12. Application Knowledge Object

The initial plan was to use Authorware as the user interface and develop the actual tutorial modules with Director and Flash and then to import the productions from Director and Flash into Authorware. One of the improvements in Authorware Attain 5 was the introduction of "Knowledge Objects" (Figure 12). Knowledge Objects are standardized templates used for common tasks in training productions. The Application Knowledge Object provided a basic user interface that is ideal for the tutorial. Figure 13 shows the interface of the modules with a legend that gives the function of each set of buttons.



Welcome to EC 38
On-line!!!

1. Next section
2. Button function window
3. Next page
4. Menu
5. Bookmarks
6. Find
7. Glossary
9. Exit

Dr. Jon Butler, Spanagel-448
831.656.3299 (O) 831.656.2760 (F)
butler@cs.nps.navy.mil
http://web.nps.navy.mil/~butler

V. P. Heuring and H. F. Jordan,
Computer Systems Design and Architecture

Figure 13. Authorware User Interface

By using Authorware as the interface, it enables the designer to maintain a consistent appearance throughout the entire tutorial. The only aspect of Figure 13 that changes throughout the tutorial is the content window; all buttons and windows of the production remain the same. Ideally, the designer must be concerned only with, the development of the content of the lessons.

The next step in developing the project was to plan and develop the tutorials. This would prove to be the most tedious and time-consuming process, partly due to some bugs in the software suite and a lack of experience.

2. Lesson Content Development

The Director development suite and Flash were used to design the lessons for Chapter 1 of the EC3840 text. These software applications were chosen for the following reasons:

- **Authorware compatibility.** Authorware advertised complete compatibility with Director and Flash. Since Authorware provides the interface for production, Director and Flash seemed to be logical choices.

- **Director development experience.** Since, I previously was exposed to the Director suite during a multimedia class, CS 3202 given by the Computer Science department, I felt that using Director would prove to be the most efficient means of developing the lessons. I assumed the updated versions would be very similar to the previous versions.

- **Director and Flash reviews.** Both programs received strong reviews from developers and evaluators. The combination of vector graphics and data streaming has made the suites attractive for development on the web.

- **Availability and price of the software.** Considering the possible initial cost of computer-based learning, as discussed previous chapters, these products were relatively affordable. In addition, the ECE department had recently purchased the Macromedia development suite. Therefore, all of the software was available on a local workstation.

- **PowerPoint import.** Director was able to import PowerPoint presentations with full functionality into productions.

a) PowerPoint Issues

As mentioned earlier, Director provides a function that allows PowerPoint presentations to be imported. This feature is attractive for the designer who is not familiar with Director or for media that has previously been developed with PowerPoint. For some novice developers, initial design may be more efficient and convenient using PowerPoint than Director. This brings two benefits to mind:

- Reusability. By implementing pre-existing or newly developed PowerPoint presentations, the developer has the ability to use the PowerPoint presentation for less intensive interactive applications.

- Director introduction. By importing media from PowerPoint, the designer can see how Director implements certain types of animations and transitions.

A question may arise, when would it be efficient to import a PowerPoint presentation into Director? As discussed previously, Director's strength is in the ability to design complex interactions. If the purpose of the PowerPoint presentation to do a linear presentation, Director would not be an ideal option.

The decision to use PowerPoint must be made at the beginning of the production. PowerPoint presentations cannot be imported into an already existing production.

The PowerPoint import is an ideal feature for individuals who are comfortable using PowerPoint or who have pre-existing media from other presentations to which they wish to add more extensive interactivity. It is my opinion, this is the most effective means of getting started with a Director production.

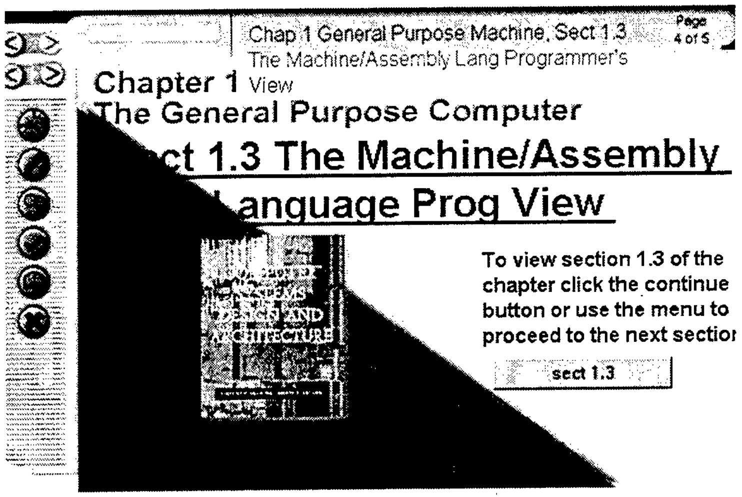
I started the lessons plan development by importing the course notes from a PowerPoint presentation. This process was relatively straightforward. I discovered that if a PowerPoint graphic was designed using one of Microsoft Office's generic drawing tools, the graphic was imported piece by piece verses the entire drawing. This turned out to make editing the drawings difficult because the imported graphics were just basic shapes, squares, lines and circles. This problem can be avoided by designing all graphics in a graphics program, like Fireworks or CorelDraw, and import them into the PowerPoint presentation. Doing this will ensure the complete graphic will be imported correctly into the Director production.

Another point worth noting is that all of the transitions and functions that can be used in PowerPoint do not import properly into Director. As an example, I was unable to successfully import into Director some of the timing features for transitions between slides from the PowerPoint presentation. This is rare; most do import without complications. I was unable to find any documentation that listed the PowerPoint features Director did not support.

b) Initial Development Issues

The initial stages of development were very tedious and time consuming. This was due to several reasons. First, I was unfamiliar with some of the features in the software I was using. I also was confused about which multimedia application would do particular jobs with the greatest ease, Flash or Director. I went from one application to another trying to establish some sort of criteria to determine the best choice for certain jobs. I believe this could have been avoided by limiting the choices of software. In retrospect, I believe the best decision would have been to only use Director for all of the multimedia development and as I became more comfortable with Director judge the feasibility of introducing other tools.

After continuing with the project and discovering more about how multimedia productions are designed using ISD, I found that many of my problems could have been avoided if I would have done a more complete analysis and pre-design planning. A consequence of this was the tutorial turned out to be more linear than I had originally planned. When I say linear, I am referring to a lack of interactivity. Because some of the problems I encountered were not anticipated, at times I felt as if I was forced me to make a decision between making major revisions or sacrificing some of the interactivity in production. The most time consuming and challenging part of the development was the development of consistent and effective interactions that are beneficial. I strongly believe that if the principles of ISD were followed the development would have proven to be more efficient and be a more effective production.

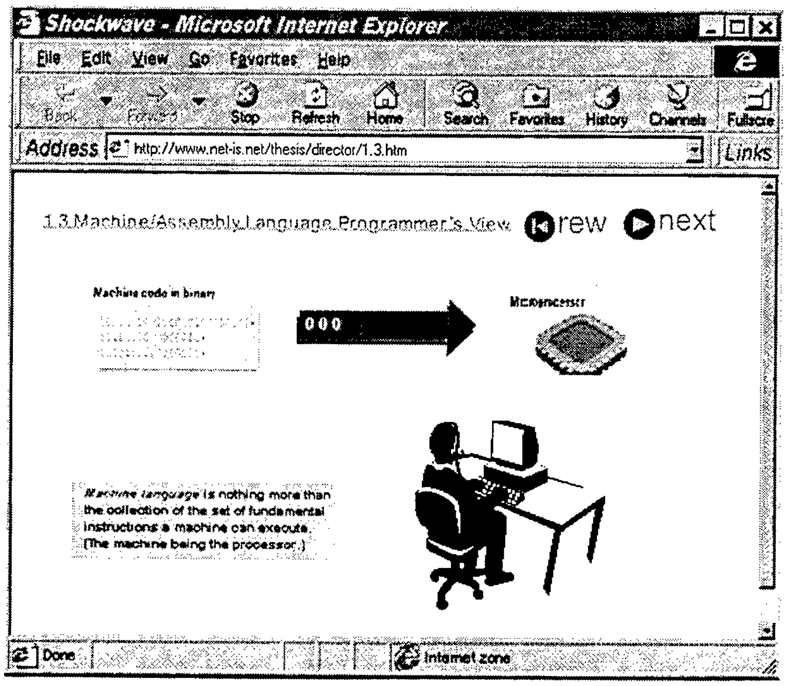

Chap 1 General Purpose Machine, Sect 1.3
The Machine/Assembly Lang Programmer's View
Page 4 of 5
Chapter 1
The General Purpose Computer
ct 1.3 The Machine/Assembly
Language Prog View
sect 1.3

To view section 1.3 of the chapter click the continue button or use the menu to proceed to the next sectio

Figure 14. Authorware Tutorial Interface

c) *General Observations*

I noticed that several times during the development of the lessons I became deeply involved using a sophisticated feature than the effective use of the tools to accomplish the lessons. That is, I felt diverted from more important parts of the production. I am not suggesting aesthetics are not an important part of a project. I believe that if the developer has a development schedule or is unfamiliar with the development tools, it is more efficient to design with the purpose of first solving the stated problem than to waste time solving problems that are not pertinent. Once again, I must emphasize that if ISD principles were used from the start of the project these types of diversion could be avoided. The "nice to do" problems can be addressed later in the development stage.


Shockwave - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscre
Address http://www.net-is.net/thesis/director/1.3.htm
Links
Machine code in binary
000
Microprocessor
rew
next
Done
Internet zone

1.3 Machine/Assembly Language Programmer's View

Machine language is nothing more than the collection of the set of fundamental instructions a machine can execute. (The machine being the processor.)

Figure 15. Sample Tutorial Screen

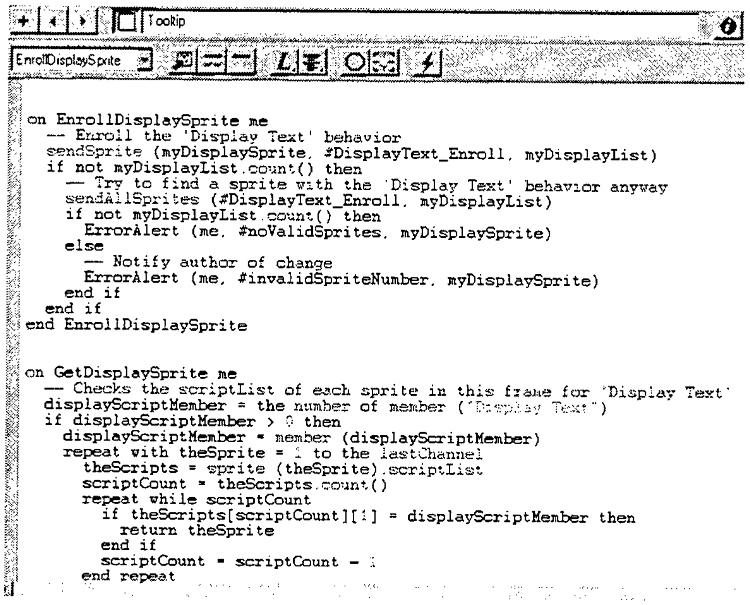
Designing in isolation and a lack of early evaluation was another problem I encountered. I believe that the quality of the lessons could have been improved significantly by consulting with a more people about ideas dealing with content and approaches to addressing certain topics of the production. The lack of early evaluation was evident when some of the modules were presented. Some problems were discovered that were fundamental to framework of production. To fix these problems additional hours of troubleshooting and development were required. Figure 15 shows the format of one of the tutorial modules.

C. PIPELINE SIMULATION DESIGN

The second aspect of the design was a tutorial explaining the concept of pipelining in a simple reduced instruction set computer. [Ref. 15] Fortunately, I had gained some valuable experience from the design of the tutorials. I attempted to implement many of the ISD principles that were not used with the tutorial production. With the pipeline production, I also decided to incorporate more scripting. Previously, I attempted to do the tutorial with as little scripting as possible. Attacking a problem with that mentality severely limits the capabilities that can be implemented.

Scripting in Lingo proved to be much efficient than graphical designing in the drag and drop mode that the tutorials were developed. Scripting provided more control over the events in the production. Object oriented programming features can be used with Lingo. Concepts such as inheritance and ancestors can be used. Developing an understanding of Lingo was well worth the time and effort spent. Figure 16 is an example of a Lingo script. Appendix

C contains all the Lingo scripts used in the development of the simulation.



```
on EnrollDisplaySprite me
  -- Enroll the 'Display Text' behavior
  sendSprite (myDisplaySprite, #DisplayText_Enroll, myDisplayList)
  if not myDisplayList.count() then
    -- Try to find a sprite with the 'Display Text' behavior anyway
    sendAllSprites (#DisplayText_Enroll, myDisplayList)
    if not myDisplayList.count() then
      ErrorAlert (me, #noValidSprites, myDisplaySprite)
    else
      -- Notify author of change
      ErrorAlert (me, #invalidSpriteNumber, myDisplaySprite)
    end if
  end if
end EnrollDisplaySprite

on GetDisplaySprite me
  -- Checks the scriptList of each sprite in this frame for 'Display Text'
  displayScriptMember = the number of member ("Display Text")
  if displayScriptMember > 0 then
    displayScriptMember = member (displayScriptMember)
    repeat with theSprite = 1 to the lastChannel
      theScripts = sprite (theSprite).scriptList
      scriptCount = theScripts.count()
      repeat while scriptCount
        if theScripts[scriptCount][1] = displayScriptMember then
          return theSprite
        end if
        scriptCount = scriptCount - 1
      end repeat
```

Figure 16. Lingo Script

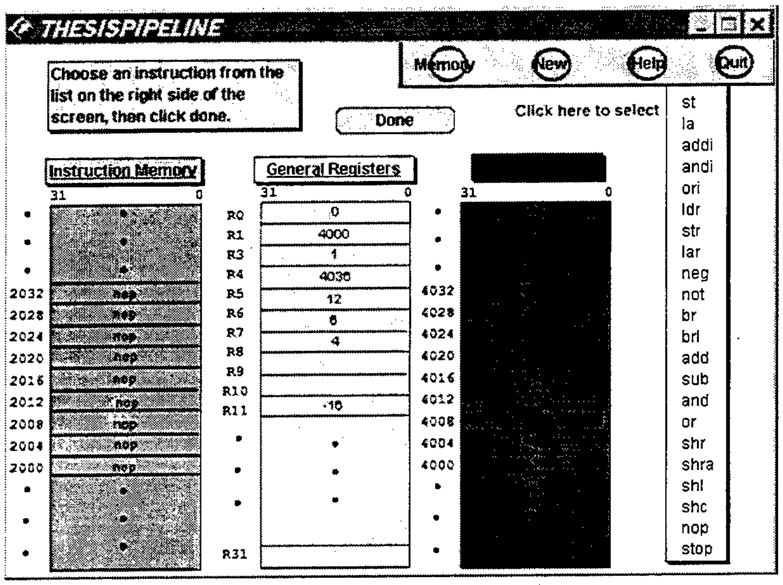
The development of this project was much quicker than the development of the tutorials. This was because I was more familiar with the development tools and followed more closely ISD principles.

The task was to simulate a five-stage pipelined RISC computer. The reference and format of the machine was taken from the EC 3840 text. [Ref. 15]

1. Initial Stages

First an analysis of the problem was conducted and a basic concept for the simulation was developed. The analysis was not as thorough as described in previous chapters, but proved to be sufficient considering time constraints.

I divided the simulation into four distinct sections. The first section would be the area where the user would choose the instruction to simulate. The user also has available the status of the all pertainent registers. Figure 17 shows the interface for the instruction choice.



Choose an instruction from the list on the right side of the screen, then click done.

	Instruction Memory
2032	nop
2028	nop
2024	nop
2020	nop
2016	nop
2012	nop
2008	nop
2004	nop
2000	nop

	General Registers
R0	0
R1	4000
R3	1
R4	4036
R5	12
R6	8
R7	4
R8	
R9	
R10	
R11	-16
R31	

Figure 17. Pipelining Simulation - Instruction Choice

The next section would allow the user to input the necessary data fields for each instruction selected. Each instruction is processed by the program and displayed as described in the EC 3840 text. The user still can view the status of the applicable registers. Figure 18 shows the interface for the fields input section of the simulation.

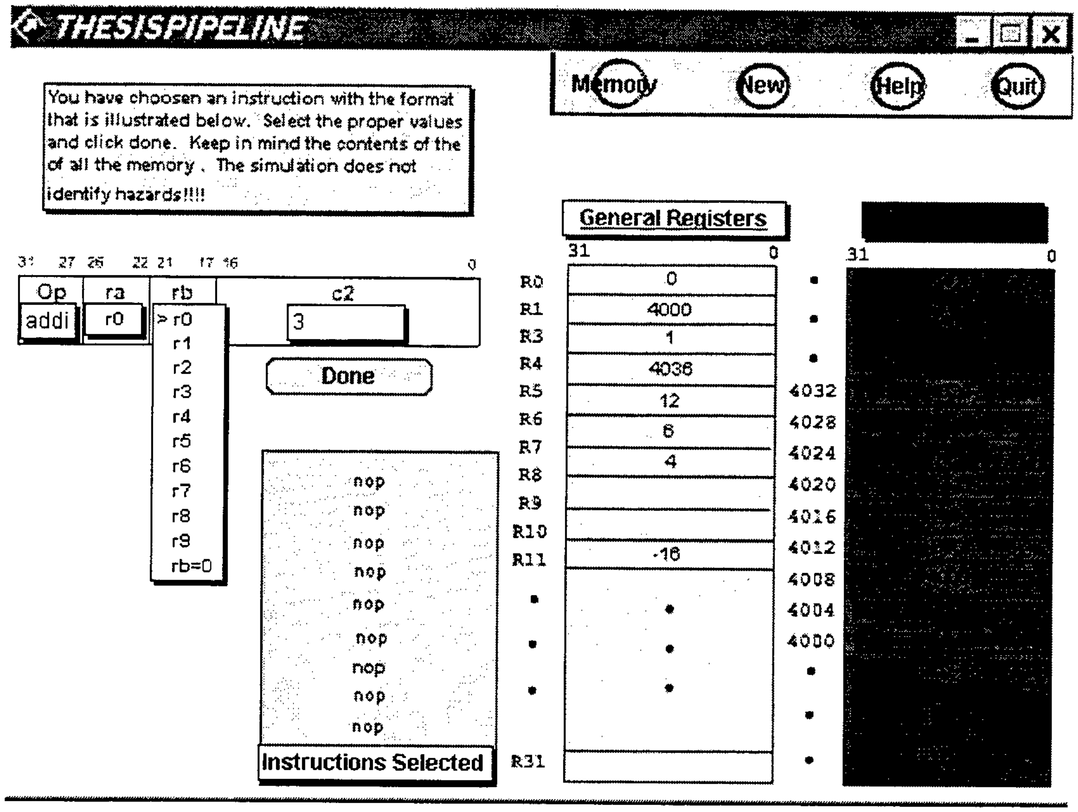

You have choosen an instruction with the format that is illustrated below. Select the proper values and click done. Keep in mind the contents of the of all the memory . The simulation does not identify hazards!!!!

Figure 18. Pipeline Simulation - Data Field Input

The next section is the schematic of the pipelined machine. The user accesses this section after he/she has selected the set of instructions to be processed by the computer. Figure 19 shows the RISC block diagram.

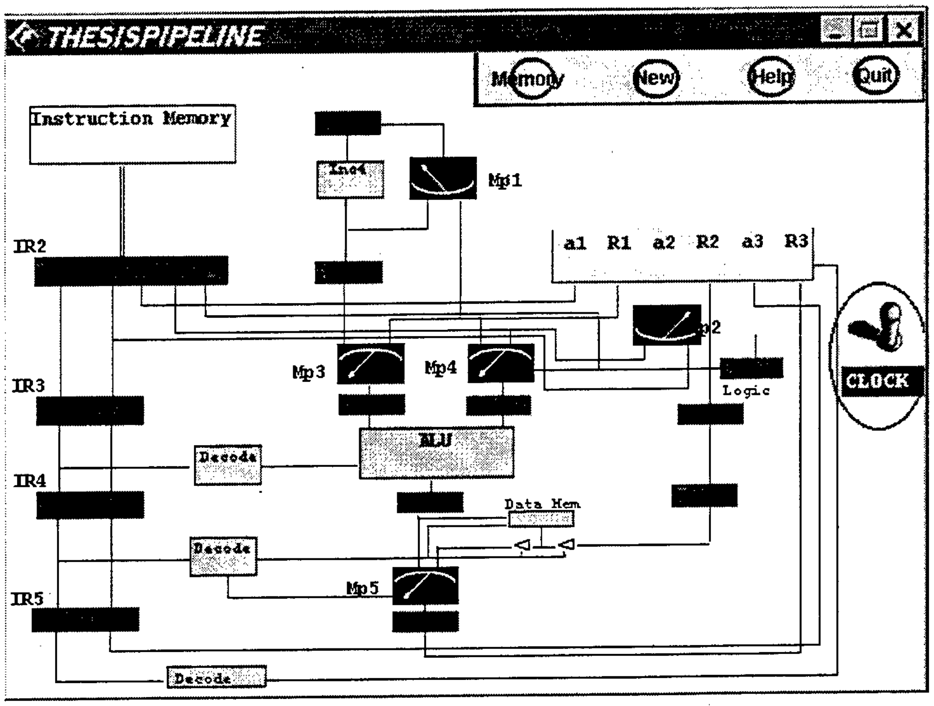


Figure 19. Pipeline Simulation - Schematic

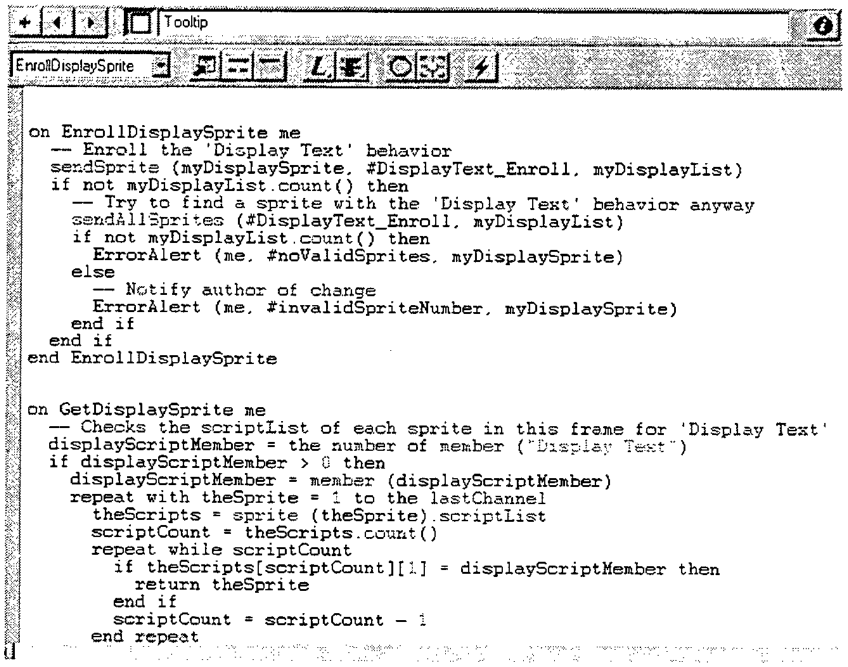
The user can view the contents of any of the registers by placing the cursor over appropriate register. The user can then "clock" the machine and observe each instruction progress through the pipeline.

2. Graphics

With this design, I concentrated on completing all the graphics and then the coding. Graphic development was straightforward. Most of the components could be drawn using basic shapes, blocks, lines and circles. I was able to use some objects that were available from the design software. An example of this, included buttons in the menu bar and the clocking switch in the pipeline block diagram.

3. Scripting

A most challenging aspect was the development of the Lingo code. I used the principles of incremental development and object oriented programming to code the scripts. The coding was divided into many different subsets of methods (small task oriented programs). The language is very similar in format to any other object oriented language. Director provides a debugger utility, a variable watch window, and a real time instruction execution window. All of these utilities were used extensively and proved to be extremely useful. Figure 20 shows a script used in the design of the simulation.



```
on EnrollDisplaySprite me
  -- Enroll the 'Display Text' behavior
  sendSprite (myDisplaySprite, #DisplayText_Enroll, myDisplayList)
  if not myDisplayList.count() then
    -- Try to find a sprite with the 'Display Text' behavior anyway
    sendAllSprites (#DisplayText_Enroll, myDisplayList)
    if not myDisplayList.count() then
      ErrorAlert (me, #noValidSprites, myDisplaySprite)
    else
      -- Notify author of change
      ErrorAlert (me, #invalidSpriteNumber, myDisplaySprite)
    end if
  end if
end EnrollDisplaySprite

on GetDisplaySprite me
  -- Checks the scriptList of each sprite in this frame for 'Display Text'
  displayScriptMember = the number of member ("Display Text")
  if displayScriptMember > 0 then
    displayScriptMember = member (displayScriptMember)
    repeat with theSprite = 1 to the lastChannel
      theScripts = sprite (theSprite).scriptList
      scriptCount = theScripts.count()
      repeat while scriptCount
        if theScripts[scriptCount][1] = displayScriptMember then
          return theSprite
        end if
        scriptCount = scriptCount - 1
      end repeat
```

Figure 20. Lingo Script

The final chapter of the study is a discussion of the findings and recommendations.

VIII. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

The undertaking of this study proved to be interesting and challenging. I discovered the potential uses of the web to aid in learning and disseminating information was extensive and only limited by the implementer's creativity. My recommendations at the end of this chapter allude to some of the possibilities. In the introduction, several questions were introduced that were to be addressed in this study. As a result of the design work undertaken, response to those questions follows.

1. Can an effective multimedia production be made using off-the-shelf software?

I believe that effective multimedia productions can be developed with today's multimedia software to supplement or replace traditional coursework. The real question is not if the software is capable, but more importantly, are the skills and resources available to use these software products effectively? There is no doubt that the software is capable of providing the designer with the tools needed to develop effective multimedia products.

Since neither of my productions have actually been evaluated or tested by any of the quantitative means discussed in this study, it is difficult to claim that my productions are effective. However, I am confident that if sound design principles are used, the multimedia products will be effective and efficient.

2. Is the development of such productions cost and time effective?

A major issue in the discussion of developing multimedia applications, is the amount of time required to become familiar with the software and then to actually implement the design. This is the most immediate of all concerns. I strongly believe that, in time, the process can be made more efficient. The only way to increase the proficiency of development is to become familiar with the tools and to use sound design principles. There was a major difference in development time when comparing the tutorial production to the pipelining production. This was due to better planning and following the guidelines described in the study when producing the pipelining section.

The development of these modules was definitely inefficient with regards to the time spent. It is unreasonable for a student or professor with no experience to believe that he/she will be able to develop a highly effective interactive course without dedicating many hours to the task. The advantage of this is, the next production will be developed more rapidly and more efficiently than the first. I also feel with adequate support or additional formal training the development time and design problems would be greatly reduced.

In regards to cost, I used the software the department purchased, which cost approximately $3000. A workstation was made available for development. This eliminated any major expense for hardware. To distribute the productions is nothing more than a matter of posting the files and extensions to a web-site. No additional cost is incurred for the user to access the material because the plugins required can be easily downloaded. I feel the actual cost is quite economical. The real judge of the effectiveness of the productions is the rate of return. That cannot be quantitatively determined from the productions I designed, but the potential is significant.

In regards to the actual time required for each revison of the EC 3840 productions, I cannot acurately quantify the amount of time spent for development. I compared some of the tasks listed in Appendix A, Estimation Guide for Development Time, with the time reqired for me to develop similar elements. The comparisions were made when I was more familiar with software tools. With that in mind, the development times in the appendix accurately reflect the time required for my development.

3. Can these systems be used in the classroom?

These types of productions definitely have a place in the classroom. The power of multimedia can definitely supplement the instruction in a classroom environment.

Simulations can make difficult concepts clear and allow for student interaction on many different levels. As the campus becomes more network oriented, I believe it will be easier to incorporate these types of productions into the classroom. These sort of learning aids will add a dimension to tradition learning. Many of the References noted in this study document successful cases of implementing multimedia-based productions into the classroom to improve on some deficiency noted in a course.

4. Can a guide of protocol be developed to aid in this study?

I believe the most important finding of this study is the benefit of ISD principles. This concept is administratively extensive, but it has proven itself to be a consistent method of developing and revising multimedia-based software. The systems engineering approach to the design of these productions can almost guarantee a successful and revisable project. With the time and effort that are required to develop these pieces, it is imperative to utilize this standardize system, so as to avoid an unusable or ineffective piece after hundreds of man-hours have been spent.

5. What kind of training is required?

It is possible to develop a production with no formal training. The problem with this approach is, with no experience, training, or on-site support, the development time will be unpredictable. Ideally, if someone is going to undertake a project, a seminar consisting of a half-day per design tool would be beneficial. It may be unreasonable to attempt to send everyone who is tasked to assist in the development of these projects to training. A definite alternative maybe, to have some individuals or even a team on staff to be the technical experts in computer-based learning systems and applications.

B. RECOMMENDATIONS

1. Supplement training for new developers.

In order to be productive quickly, training is necessary. The software alone is intimidating and sometimes frustrating. A lack of understanding about the design strategies can also be addressed during training. The main point is that technology is only the vehicle to achieve certain learning objectives.

2. For new developers or developers with media presentations already using PowerPoint, import into Director.

Given that Director was the multimedia development tool choice, the most effective method to start a production is through the use of importing a PowerPoint presentation. Many of the transitions used in PowerPoint function without further modification in Director.

3. Sponsor a coordinated effort across departments to research projects and focus on possible solutions.

Many NPS departments are experimenting with different uses of the web, the school's Intranet, and other computer-based learning applications. I recommend a team consisting of students, faculty, and consultants examine in depth the feasibility of technology-based learning. This study should encompass all departments to utilize their specific expertise. I believe we have much of the knowledge base already in place but lack a conserted effort.

4. Choose a test curriculum or class to fully implement computer-based training.

The major focus of my study was to develop a module and a simulation. This can be beneficial to students and professors. I feel that the true power of technology-based learning comes with the implementation of a larger scale project. I realize that a professor or student cannot be expected to develop an entire course and still have time to accomplish other duties. It might be more reasonable to contract the production and use the professors and faculty as the content technical experts. The reason I recommend this is the effectiveness of technology-based learning is directly tied to the rate of return. The easiest and most impressive measure of that rate is monetary. For example, suppose some of the introductory courses (2000 level) could be offered via the web or even via CD. Students could even have access to the productions before they arrive at NPS. This may possibly save thousands of dollars over the life of production. Class could only meet once a week or even never. In the latter case, students proceed at their own pace and validate the cause by taking a test when they feel comfortable with the material. Since the technology of basic calculus or circuit analysis rarely change, the production could continue to produce a return indefinitely.

Despite the fact that some tutorials have been developed for EC 3840, I recommend future development of an entire course, be targeted at a 2000 level course such as introductory circuits or digital logic. The reason for this recommendation is EC 3840 content is not standard. The notes and text cannot be assumed to be constant throughout the life of the course. Considering that the lesson plans and tutorial are based on the current text, a change of text could result in parts of the production becoming irrelevant. I do recommend the development of productions targeted at particular concepts of EC 3840 (i.e. pipelining, RTN), not the entire course, that remain constant in the study of computer architecture.

5. Provide a department graphics and media directory.

A time consuming aspect of developing these productions is the development of graphics. As productions or presentations are developed, all graphics incorporated in the design should be cataloged and retained for future projects. The type of graphical elements I am referring to are electronic components, buttons and other graphical objects that are commonly used in the department.

6. Examine the feasibility of developing solutions with alternative programs.

My focus was on using the Macromedia suite. There are many other vendors, which provide other products with similar features.

7. Faithfully use the ISD principles.

This is key if the productions are to be designed in an efficient manner and be able to be revisable. Be especially aware of the need to fully document the development procedure.

8. Examine the possibility of implementing some other technical solution.

Possibly, multimedia-based software development may not be feasible at this time. Examine the possibilities of alternative approaches to utilize technology to enhance the learning experience. For example, Stanford University has a computer-based learning program where every lecture has been video taped and digitized. The lectures are broadcast at scheduled times over the web and the students post messages to each other and the instructor via email. Possibly, some classes can be offered with an alternative similar Stanford.

C. FINAL COMMENTS

Computer-based learning is a powerful tool that, when developed properly, can be effective and accomplished in an economical and efficent manner. The key to that statement is the assumption that the training is developed properly, using instructional architectures and ISD principles.

Software tools used to develop multimedia-based instructional modules should be choosen regarding user expertise and available support for the application. If an inexperienced designer is expected to be productive as soon as possible, training will be required. In addition, support for each of the applications should be made available. Examples include telephone or online technical support or possibly an in-house expert.

If the decision to continue development of computer-based learning modules is made, choose a course whose content remains constant over a long enough time to recover development costs. Since the development of these productions are time and cost extensive, it is important to attempt to receive the largest rate of return on each production. The productions used for these types of classes will have the longest periods of usability thus, increasing the rate of return.

Computer-based learning is definitely an area of study that requires more research. The potential benefits to students and faculty are numerous. The possible financial benefits in the long term are another reason computer-based learning is attractive. In contrast, the development of these systems can be costly and ineffective if sound design principles are not implemented. That is the reason I recommend continuing research in methods of developing and implementing computer-based learning systems at NPS.

APPENDIX A. ESTIMATION GUIDE FOR DEVELOPMENT TIME

Type of Element	**Complexity of Element**	**Estimated Development Time**	**# of these Element**	**Total element time**
Presentation elements (screen with text, video clips, animations, etc. that do not require any student input or response analysis)				
	Straight text screen	15 min/screen		
	Text w/ graphics	20 min/screen		
	Text w/ audio, video, or animations	30 min/screen		
Media Assets (clips or graphic)				
	Simple graphic	30 min		
	Complex illustration	4 hrs		
	Audio clip	1 hr/clip		
	2-D animation	1 hr/animation		
	3-D animation	1 day/animation		
	"Talking head" video	30 min/video min		
	Documentation of demonstration of real life scene (i.e. shot of actual factory floor)	2 hrs/ea location		
	Dramatization	4 hrs/scene		
	Virtual reality model	5 days/ea model		
Interaction Element				
Type of Element	**Complexity of**	**Estimated**	**# of**	**Total**

	Element	**Development Time**	**these Element**	**element time**
	Simple hyperlink, menu, or yes/no question with branching to another screen	30 min/question or hyperlink		
	Multiple choice question w/ feedback	1 hr/question		
	Fill-in question with multiple feedback for different responses	2 hr/question		
	Complex analysis (more than one response per question or branching based on an analysis of response patterns to more than one question)	3 hrs/question		
	Simulation responses with multiple variables and complex feedback	1 day/interaction		
Total time			Subtotal	
Factors that increase development time			**% increase**	**+ days**
	Content totally new and in transition		+60	
	Content is not new, but has never been taught before		+40	
	Content is controversial or difficult to specify exactly what's right or wrong		+30	

	Sponsor's first use of technology		+30	
	Sponsor has had bad experience w/ technology previously		+20	
	More than three people are responsible for approvals and/or providing content		+40	
	Content experts or approvers not easily accessible by developers		+20	
	Content or media elements need to be licensed or acquired through permission from some third party		+30	
	Designers have never written an interactive learning program		+100	
	Developers are using unfamiliar authoring tools		+30	
	Developers are unfamiliar with playback hardware system or there are different hardware specs for the target audience		+20	
	Hardware or authoring tools are newly released or beta versions		+60	
Factors that increase development time			**% increase**	**+ days**

	Writers are unfamiliar with content		+30	
	The development team is working on more 2 projects		+20	
Total extra days project complexity factors				
Plus subtotal from above				
Grand total project person days				

Note: "Person-days" Refers to the total project effort, not the actual length of time the project will take. For example, a project might encompass 300 person days, but be accomplished by a project team in 60 calendar days.

Table taken from [Ref. 1]

APPENDIX B. COST ANALYSIS DATA SHEETS

Instructor-led Training vs. TBT

Delivery cost comparison chart Item	**Instructor -led**	**TBL**
Instructor costs X # of instructors • Preparation time • Salary during travel time X # of instructors • Airline costs • Lodging costs • Meal costs • Rental car costs		
Delivery sites • Data communications Computers Fax Machines • Equipment • Facilitation support • Material distribution • Network (video/audio computer) Installation Usage • Room cost • Telephone		
Totals		

[Ref. 6]

ROI Analysis in Four Steps

1. Identifying the present costs:

Is this topic currently being taught?

If yes… What is your estimate of the current total cost for this instruction? Include not only the cost of designing and duplicating materials, but also the cost of travel and time of presenters and attendees.

____

If no… What do you think it costs your organization not to have training in this area?

____

2. Estimating the infrastructure costs of a computer-based solution:

If you are doing this even partly in-house, what is your estimate for purchasing the necessary hardware/software for authoring and updating? As a rule-of-thumb, estimate $10,000 per authoring station. If you already have the hardware and software, or are doing this completely out-of-house, just enter "no cost".

____

What is your estimate for purchasing the necessary hardware for playback? As a rule-of-thumb, estimate $2,500 for a multimedia equipped PC. If you already have the necessary playback hardware, just write "no cost". Be sure to calculate the cost for the total number of workstations necessary.

____

3. Estimate the cost savings:

If you already have training on this topic:

How much will you save in travel if you use computer-mediated learning?

____

How much will you save in terms of salaries of learners given that learning will be more efficient and travel is eliminated?

____

What costs will be reduced given that learning can happen more quickly and can be individually scheduled?

____

If you don't have any training on this topic:

How much will effective and timely training save in terms of avoiding mistakes, capitalizing on opportunities, etc.?

____

Given the information above, how much can you spend per program, including amortization of the equipment and show a return on investment?

____

[Ref. 1]

APPENDIX C. DIRECTOR SCRIPTS AND CAST USED IN PIPELINE SIMULATION

```
on startMovie

  global InstMemList, NumInst
  InstMemList = [[instType:8, instAddress:2000, op:"nop"],¬
[instType:8, instAddress:2004, op:"nop"],¬
[instType:8, instAddress:2008, op:"nop"],¬
[instType:8, instAddress:2012, op:"nop"],¬
[instType:8, instAddress:2016, op:"nop"],¬
[instType:8, instAddress:2020, op:"nop"],¬
[instType:8, instAddress:2024, op:"nop"],¬
[instType:8, instAddress:2028, op:"nop"],¬
[instType:8, instAddress:2032, op:"nop"],¬
[instType:8, instAddress:2036, op:"nop"],¬
[instType:8, instAddress:2040, op:"nop"],¬
[instType:8, instAddress:2044, op:"nop"],¬
[instType:8, instAddress:2048, op:"nop"],¬
[instType:8, instAddress:2052, op:"nop"],¬
[instType:8, instAddress:2056, op:"nop"]]
  initInstFetch
  NumInst = 0
  memInit
  genRegInit


  --initsim



end startMovie


-- DESCRIPTION --

on getBehaviorDescription me
  return "¬
HOLD ON CURRENT FRAME"&RETURN&RETURN&"¬
Drop this behavior into the Script Channel of the Score or onto the
Stage ¬
in order to keep the playback head in the current
frame."&RETURN&RETURN&"¬
PARAMETERS: None"
end getBehaviorDescription

on getBehaviorTooltip me
  return "¬
Frame behavior."&RETURN&RETURN&"¬
Holds the playback head still."
end getBehaviorTooltip



-- HISTORY --

-- 3 November, written for the D7 Behaviors Palette by James Newton
```

```
on exitFrame me
  go the frame
end exitFrame


-- DESCRIPTION --

on getBehaviorDescription me
  return "¬
RADIO BUTTON GROUP"&RETURN&RETURN&"¬
Select a group of radio button sprites and drop this behavior onto one
of ¬
them.  You will be asked to give a unique ID to the group  so that its
¬
members can recognise each other."&RETURN&RETURN&"¬
When the movie is running, switching one button in the group ON will
switch ¬
all the others OFF.  By default, the button in the lowest sprite channel
will ¬
be ON.  You can however set which button is on by default by opening ¬
the Behavior Parameters dialog for that button and setting 'Default
status of ¬
button: on'.  If more than one button is selected as default, then the ¬
'default' button in the highest channel will be chosen.  All other
buttons in ¬
the same group will be switched off."&RETURN&RETURN&"¬
If you wish the behavior to preserve the radio button selection then
choose ¬
'Default status of button: auto' for ALL the buttons in the group.  This
will ¬
prompt the behavior to convert the hilited button to a checkBox on
endSprite.  ¬
When the user returns to the section containing the radioButton group,
the ¬
behavior will set the checkBox back to a radioButton, and use that
button as ¬
the default hilite.  If you save the cast containing the buttons at the
end of ¬
the session, the default values will even be preserved when the user
quits ¬
your application."&RETURN&RETURN&"¬
The behavior returns the currently selected button in response to a call
with ¬
the following syntax:"&RETURN&RETURN&"¬
  sendAllSprites (#RadioGroup_SelectedButton,
'UniqueID')"&RETURN&RETURN&"¬
See the 'Notes for Developers' in the script itself for more details."&¬
RETURN&RETURN&"¬
PERMITTED MEMBER TYPE:"&RETURN&"Button members"&RETURN&RETURN&"¬
PARAMETERS:"&RETURN&RETURN&"¬
* Radio group ID"&RETURN&RETURN&"¬
Give the same unique ID to all members of a group.  The easiest way to
do ¬
this is to select all the Radio Buttons in the Score or on the Stage,
then ¬
drop this behavior onto one of them.  The Behavior Parameters dialog
will open ¬
only once and all sprites will receive the same initial parameters."&¬
RETURN&RETURN&"¬
* Button acts as default for the group (TRUE | FALSE)"&RETURN&RETURN&"¬
```

```
Using a unique ID allows you to have more than one Radio Button group on
the ¬
Stage at any one time, each acting independently of the
others"&RETURN&RETURN&"¬
PUBLIC METHODS:"&RETURN&"¬
=> Obtain the name, number and sprite number ¬
of the selected button in the group"&RETURN&RETURN&"¬
ASSOCIATED BEHAVIOR:"&RETURN&"¬
+ Multistate Toggle Button"&RETURN&"¬
Gives you similar features using graphic members."
end getBehaviorDescription



on getBehaviorTooltip me
  return " ¬
Drop this behavior on a group of Radio Buttons to ensure"&RETURN&"¬
that only one button in the group can be on at a time."&RETURN&RETURN&"¬
You can create several independent Radio Button groups"&RETURN&"¬
on the Stage.  To do so, give each group a unique ID."&RETURN&RETURN&"¬
Place the default selected button in the lowest channel."&RETURN&"¬
or select a specific button in the group to act as default."
end getBehaviorTooltip



-- NOTES FOR DEVELOPERS --

-- This behavior seeks out other sprites with the same behavior.   Any
sprite
-- with the same ID is added to a shared list: ourGroupList.  Each
behavior in
-- the group holds a pointer to this shared list.
--
-- Selecting one button in the group triggers a RadioGroup_Toggle call
to
-- to all the behaviors in ourGroupList.  This sets the hilite of all
other
-- buttons in the group to FALSE.  (The selected button ignores its own
call).

-- TESTING WHICH BUTTON IS SELECTED
-- To determine which button in a given Radio Group is currently
selected, you
-- can type the following in the Message window:
--
--   put sendAllSprites (#RadioGroup_SelectedButton, "Group name")
--
-- If a button is selected, this will return a property list of the
following
-- form:
--
--   [#name: "buttonName", #number: <number in group>, #sprite:
<spriteNum>]
--
-- If you want to know which button is selected in several Radio Groups,
use an
-- empty property list instead of a Group name:
--
--   put sendAllSprites (#RadioGroup_SelectedButton, [:])
--
```

```
-- This will return a nested property list of the form:
--
--   ["Group 1": [#name: "x", #number: y, #sprite: z], "Group 2": [...]]

-- TIP: You could also interrogate the Radio Button members
individually.  Try:
--
--   put member ("Default btn").hilite
-- OR
--   put sprite(1).member.hilite

-- TROUBLESHOOTING
-- Symptom: When you halt the movie, the hilited button converts to a
checkBox
-- button.
-- This is intended.  It means that you have selected the 'auto' default
mode
-- in the Behavior Parameters dialog.  This converts the currently
hilited
-- button to a checkBox when you halt the movie, or when the user leaves
the
-- current section.  The next time the user returns to the section, the
¬
-- behavior uses the buttonType property to detect which button was
formerly
-- hilited.  If you place the radioButtons in an external cast, you can
use the
-- "save CastLib" command to save the new state of the buttons.  This
will
-- preserve the settings even when the user quits your application.



-- HISTORY --

-- 11 September 1998: written for the D7 Behaviors Palette by James
Newton
-- 25 October   1998: descriptions revised, not-button member alert
fixed
--  3 November  1998: allowed selection of a default button
-- 17 November  1998: added #on|#off|#auto default modes



-- PROPERTIES --

property spriteNum
-- author-defined parameters
property myDefault
-- shared properties
property ourID          -- string common to all buttons in a group
property ourGroupList   -- list of behaviors in the group
-- internal properties
property mySprite
property myButtonMember
property myName         -- Text of button member (or its name or member
ref)



-- EVENT HANDLERS --
```

```
on beginSprite me
  Initialize me
end beginSprite


on mouseUp me
  Toggle me
end mouseUp


on endSprite me
  SaveSetting me
end endSprite



-- CUSTOM HANDLERS --

on Initialize me -- sent by beginSprite
  mySprite       = sprite(me.spriteNum)
  myButtonMember = mySprite.member
  memberType     = myButtonMember.type

  -- Error checking
  case memberType of
    #button: -- do nothing
    otherwise
      ErrorAlert (me, #invalidMember, memberType)
  end case
  -- End of error checking

  if myButtonMember.buttonType <> #radioButton then
    myButtonMember.buttonType = #radioButton
    myButtonMember.hilite = TRUE
  else
    myButtonMember.hilite = FALSE
  end if

  -- Determine the button's name
  myName      = myButtonMember.text
  if myName   = EMPTY then
    myName    = myButtonMember.name
    if myName = EMPTY then
      myName  = myButtonMember.member
    end if
  end if
  -- (Re)create the shared ourGroupList
  ourGroupList = []
  sendAllSprites (#RadioGroup_RollCall, ourID, ourGroupList)

  -- Ensure that the lowest button is ON if none have been chosen
  if ourGroupList.count() = 1 then
    myButtonMember.hilite = TRUE
  else
    case myDefault of
      #on:
        call (#RadioGroup_Toggle, ourGroupList, FALSE)
        myButtonMember.hilite = TRUE
      #auto:
        if myButtonMember.hilite then
          call (#RadioGroup_Toggle, ourGroupList, FALSE)
          myButtonMember.hilite = TRUE
        end if
    end case
  end if
end Initialize


on Toggle me -- sent by mouseDown, CheckForRollover
  -- Toggles the current button on, and all others off...
  if myButtonMember.hilite then
    call (#RadioGroup_Toggle, ourGroupList, FALSE, me)
  else
    --...or keeps the current button ON if it was already
    myButtonMember.hilite = TRUE
  end if
end Toggle


on SaveSetting me
  if myDefault = #auto then
    if myButtonMember.hilite then
      -- Change the buttonType of the button to indicate its hilite
state
      myButtonMember.buttonType = #checkBox
    end if
  end if
end SaveSetting


-- INTER-SPRITE COMMUNICATION (responses to #sendAllSprites, #call) --

on RadioGroup_RollCall me, groupID, groupList
  -- sent to all sprites by each new button on Initialize
  if groupID = ourID then
    ourGroupList = groupList
    ourGroupList.append(me)
  end if
end RadioGroup_RollCall


on RadioGroup_Toggle me, whichState, callingBehavior
  -- called by other sprites in the group on Initialize, Toggle
  -- Switches the current button to whichState if it isn't already
  if (callingBehavior = me) then exit
  if (myButtonMember.hilite = whichState) then exit

  myButtonMember.hilite = whichState
end RadioGroup_Toggle



-- PUBLIC METHODS (responses to #sendSprite, #sendAllSprites, #call) --

on RadioGroup_SelectedButton me, propListOrString
  -- returns list giving name, number and spriteNumber of button (if
selected)
  if not myButtonMember.hilite then exit
  -- Is this a call to a named group or a general call using a property
list?
  if stringP (propListOrString) then
    if propListOrString <> ourID then
      -- Call intercepted by a member of the wrong group
      exit
    end if
  end if

  groupNumber = ourGroupList.getPos(me)
  data = [:]
  data.addProp (#name, myName)
  data.addProp (#number, groupNumber)
  data.addProp (#sprite, spriteNum)

  if ilk (propListOrString) <> #propList then
    return data
  else
    propListOrString.addProp(ourID, data)
    return propListOrString
  end if
end RadioGroup_SelectedButton



-- ERROR CHECKING --

on ErrorAlert me, theError, data
  -- sent by getPropertyDescriptionList, Initialize
  -- Warns the author if the behavior is attached to the wrong member
type.
  -- Determine the behavior's name
  behaviorName = string (me)
  delete word 1 of behaviorName
  delete the last word of behaviorName
  delete the last word of behaviorName
  case ilk(data) of
    #symbol: data = "#"&data
  end case


  case theError of
    #getPDLError:
      beep
      return ¬
[ ¬
 #getPDLError: ¬
 [ ¬
  #comment: "ERROR:  Click 'Cancel'.   This behavior"&RETURN&¬
                     "only works with Button Members", ¬
  #format:  #string, ¬
  #range:   [""], ¬
  #default:  "" ¬
 ] ¬
]

    #invalidMember:
      alert "¬
BEHAVIOR ERROR: Frame "&the frame&", Sprite
"&me.spriteNum&RETURN&RETURN&"¬
```

```
Behavior "&behaviorName&" only functions with Button
Members."&RETURN&RETURN&"¬
Current member type = "&data
      halt

  end case
end ErrorAlert



-- AUTHOR-DEFINED PARAMETERS --

on getPropertyDescriptionList me

  if not the currentSpriteNum then exit

  -- Error check: does the current sprite contain a #button member?
  theMember = sprite(the currentSpriteNum).member
  memberType = theMember.type
  if theMember.type <> #button then
    return ErrorAlert (me, #getPDLError)
  end if

  return ¬
[ ¬
 #ourID: ¬
 [ ¬
  #comment: "ID string for the radio button group:", ¬
  #format:  #string, ¬
  #default:  "RadioGroup "&the currentSpriteNum ¬
 ], ¬
 #myDefault: ¬
 [ ¬
  #comment: "Default status of button:"&RETURN&¬
            "(may be overriden: see notes).", ¬
  #format:  #symbol, ¬
  #range:  [#off, #on, #auto], ¬
  #default: #off ¬
 ] ¬
]
end getPropertyDescriptionList



on mouseUp
  global instMemList


  moreInst = sendAllSprites (#RadioGroup_SelectedButton,
"moreInstButtons")
  if (moreInst[#name] = "Choose another instruction") then

    -- get the fields for the instruction
    opCodeSelected = sendAllSprites (#DropList_Selection,
"opCodeSelectList")
    getUserInstChoice opCodeSelected[#text]
  else
--    load15Inst
    initSim
  end if
```

```
end


on initSim
  global controlSignals, switch, PC, logic, ¬
stage2Regs, stage3Regs, stage4Regs, Z5, IR, InstMemList, NumInst,regFile
  --  load15Inst
  initRegs
  initControlSignals
  initLogic
  initSwitches
  go to frame "sim"
  SendSprite (11, #Tooltip_SetMessage,"PC =" &&  PC)
  SendSprite (25, #Tooltip_SetMessage, "Inc4 =" && logic[#inc4])
  SendSprite (6, #Tooltip_SetMessage, "Mp1 =" && switch[#Mp1])
  SendSprite (12, #Tooltip_SetMessage, "PC2 =" && stage2Regs[#PC2])
  SendSprite (22, #Tooltip_SetMessage, "op =" && IR[2][#op] && "ra =" &&
IR[2][#ra]¬
 && "rb =" && IR[2][#rb] && "rc =" && IR[2][#rc] && "c1 =" && IR[2][#c1]
&&¬
 "c2 =" && IR[2][#c2] && "c3 =" && IR[2][#cond] && logic[#shiftCount])
  SendSprite (24, #Tooltip_SetMessage, "a1 =" && regFile[#a1] && "R1 ="
&& regFile[#R1]¬
 && "a2 =" && regFile[#a2] && "R2 =" && regFile[#R2] && "a3 =" &&
regFile[#a3]¬
 && "R3 =" && regFile[#R3])
  SendSprite (9, #Tooltip_SetMessage, "Mp2 =" && switch[#Mp2])
  SendSprite (8, #Tooltip_SetMessage, "Mp4 =" && switch[#Mp4])
  SendSprite (7, #Tooltip_SetMessage, "Mp3 =" && switch[#Mp3])
  SendSprite (15, #Tooltip_SetMessage, "Branch logic =" &&
logic[#branch])
  SendSprite (14, #Tooltip_SetMessage, "MD3 =" && stage3Regs[#MD3])
  SendSprite (4, #Tooltip_SetMessage, "Y3 =" && stage3Regs[#Y3])
  SendSprite (13, #Tooltip_SetMessage, "X3 =" && stage3Regs[#X3])
  SendSprite (28, #Tooltip_SetMessage, "ALU =" && logic[#alu])
  SendSprite (21, #Tooltip_SetMessage, "op =" && IR[3][#op] && "ra =" &&
IR[3][#ra])
  SendSprite (20, #Tooltip_SetMessage, "op =" && IR[4][#op] && "ra =" &&
IR[4][#ra])
  SendSprite (19, #Tooltip_SetMessage, "op =" && IR[5][#op] && "ra =" &&
IR[5][#ra])
  SendSprite (44; #Tooltip_SetMessage, "op =" && IR[3][#op])
  SendSprite (27, #Tooltip_SetMessage, "op =" && IR[4][#op])
  SendSprite (18, #Tooltip_SetMessage, "Z5 =" && Z5)
  SendSprite (10, #Tooltip_SetMessage, "Mp5 =" && switch[#Mp5])
  SendSprite (16, #Tooltip_SetMessage, "MD4 =" && stage4Regs[#MD4])



end

on simCont
  global controlSignals, switch, PC, logic, ¬
stage2Regs, stage3Regs, stage4Regs, Z5, IR, InstMemList, NumInst,
regFile

  loadIRs
  loadPC
```

```
   getInc4
   getMp1
   loadPC2
   loadStage4Regs
   loadStage3Regs
   getControlSignalList
   getLogic
   getSwitches
   SendSprite (11, #Tooltip_SetMessage,"PC =" &&  PC)
   SendSprite (25, #Tooltip_SetMessage, "Inc4 =" && logic[#inc4])
   SendSprite (6, #Tooltip_SetMessage, "Mp1 =" && switch[#Mp1])
   SendSprite (12, #Tooltip_SetMessage, "PC2 =" && stage2Regs[#PC2])
   SendSprite (22, #Tooltip_SetMessage, "op =" && IR[2][#op] && "ra =" &&
IR[2][#ra]¬
 && "rb =" && IR[2][#rb] && "rc =" && IR[2][#rc] && "c1 =" && IR[2][#c1]
&&¬
 "c2 =" && IR[2][#c2] && "c3 =" && IR[2][#cond] && logic[#shiftCount])
   SendSprite (24, #Tooltip_SetMessage, "a1 =" && regFile[#a1] && "R1 ="
&& regFile[#R1]¬
 && "a2 =" && regFile[#a2] && "R2 =" && regFile[#R2] && "a3 =" &&
regFile[#a3]¬
 && "R3 =" && regFile[#R3])
   SendSprite (9, #Tooltip_SetMessage, "Mp2 =" && switch[#Mp2])
   SendSprite (8, #Tooltip_SetMessage, "Mp4 =" && switch[#Mp4])
   SendSprite (7, #Tooltip_SetMessage, "Mp3 =" && switch[#Mp3])
   SendSprite (15, #Tooltip_SetMessage, "Branch logic =" &&
logic[#branch])
   SendSprite (14, #Tooltip_SetMessage, "MD3 =" && stage3Regs[#MD3])
   SendSprite (4, #Tooltip_SetMessage, "Y3 =" && stage3Regs[#Y3])
   SendSprite (13, #Tooltip_SetMessage, "X3 =" && stage3Regs[#X3])
   SendSprite (28, #Tooltip_SetMessage, "ALU =" && logic[#alu])
   SendSprite (21, #Tooltip_SetMessage, "op =" && IR[3][#op] && "ra =" &&
IR[3][#ra])
   SendSprite (20, #Tooltip_SetMessage, "op =" && IR[4][#op] && "ra =" &&
IR[4][#ra])
   SendSprite (19, #Tooltip_SetMessage, "op =" && IR[5][#op] && "ra =" &&
IR[5][#ra])
   SendSprite (44, #Tooltip_SetMessage, "op =" && IR[3][#op])
   SendSprite (27, #Tooltip_SetMessage, "op =" && IR[4][#op])
   SendSprite (18, #Tooltip_SetMessage, "Z5 =" && Z5)
   SendSprite (10, #Tooltip_SetMessage, "Mp5 =" && switch[#Mp5])
   SendSprite (16, #Tooltip_SetMessage, "MD4 =" && stage4Regs[#MD4])


end


on clearInstMem
  member("2000").text = "nop"
  member("2004").text = "nop"
  member("2008").text = "nop"
  member("2012").text = "nop"
  member("2016").text = "nop"
  member("2020").text = "nop"
  member("2024").text = "nop"
  member("2028").text = "nop"
  member("2032").text = "nop"
end


on initControlSignals
```

```
  global controlSignals
  controlSignals = ["","","","",""]
  repeat with i = 5 down to 2
    controlSignals[i] = getControlSignals (8, "nop")
  end repeat
end



on getControlSignals (instType, OpCode)


  --set up control code equations
  --initialize control signals
  set opCodeList = [#ld:false, #st:false, #la:false, #addi:false,
#andi:false, #ori:false, #ldr:false, #str:false, #lar:false, #neg:false,
#not:false, #br:false, #brl:false, #addd:false, #sub:false, #andd:false,
#orr:false, #shr:false, #shra:false, #shl:false, #shc:false, #nop:false,
#stopp:false]
  set controlCodeList = [branch:false, sh:false, alu:false, imm:false,
load:false, ladr:false, store:false, ls:false, regWrite:false,
dsp:false, rl:false, cond:false]


  --set opCode
  case OpCode of
    "add":opCodeList[#addd] = true
    "and":opCodeList[#andd] = true
    "or":opCodeList[#orr] = true
    otherwise:
      opCodeList[OpCode] = true
  end case

  --set cond code signal

  if (instType = 4 or instType = 5) then
    controlCodeList[#cond] = true
  end if

  if (instType = 7) then
    countField = true
  end if


  --  control signals defined
  if opCodeList[#br] or opCodeList[#brl] then
    controlCodeList[#branch] = true
  end if

  if (opCodeList[#shr] or opCodeList[#shra] or opCodeList[#shl] or
opCodeList[#shc]) then
    controlCodeList[#sh] = true
  end if

  if (opCodeList[#addd] or opCodeList[#addi] or opCodeList[#sub] or
opCodeList[#neg] or opCodeList[#andd] or opCodeList[#andi] or
opCodeList[#orr] or opCodeList[#ori] or opCodeList[#not] or
opCodeList[#sh]) then
    controlCodeList[#alu] = true
  end if
  if (opCodeList[#addi] or opCodeList[#andi] or opCodeList[#ori] or
(opCodeList[#sh] and not(countField))) then
    controlCodeList[#imm] = true
  end if

  if (opCodeList[#ld] or opCodeList[#ldr]) then
    controlCodeList[#load] = true
  end if

  if (opCodeList[#la] or opCodeList[#lar]) then
    controlCodeList[#ladr] = true
  end if

  if (opCodeList[#st] or opCodeList[#str]) then
    controlCodeList[#store] = true
  end if

  if (controlCodeList[#load] or controlCodeList[#ladr]  or
controlCodeList[#store]) then
    controlCodeList[#ls]    = true
  end if

  if (controlCodeList[#load] or controlCodeList[#ladr] or
controlCodeList[#alu] or opCodeList[#brl]) then
    controlCodeList[#regWrite] = true
  end if

  if (opCodeList[#ld] or opCodeList[#st] or opCodeList[#la]) then
    controlCodeList[#dsp]     = true
  end if

  if (opCodeList[#ldr] or opCodeList[#str] or opCodeList[#lar]) then
    controlCodeList[#rl]      = true
  end if


  return controlCodeList

end


on getControlSignalList
  global controlSignals, IR

  repeat with i = 4 down to 2
    controlSignals[i] = getControlSignals (IR[i][#instType], IR[i][#op])
  end repeat

end

on initLogic
  global logic, IR

  logic = [inc4:2004, branchLogic:IR[2][#cond], alu:"", decode4:"",
decode5:""]

end
```

```
on getLogic
  global logic

  --getInc4
  getBranchLogic
  getALU
  getDecode4
  getDecode5

end


on getInc4
  global logic, PC

  logic[#inc4] = PC + 4
end


on getBranchLogic
  global logic, controlSignals
  logic[#branchLogic] = controlSignals[2][#cond]
end

on getALU
  global logic, IR, stage3Regs, controlSignals

  if controlSignals[3][#ls] = true then
    logic[#alu] = stage3Regs[#X3] + stage3Regs[#Y4]
  else if controlSignals[3][#brl] = true then
    logic[#alu] = stage3Regs[#X3]
  else if controlSignals[3][#alu] = true then
    case (IR[3][#op]) of
      "add", "addi":
        logic[#alu] = stage3Regs[#X3] + stage3Regs[#Y4]
      "sub":
        logic[#alu] = stage3Regs[#X3] - stage3Regs[#Y4]
      "not":
        logic[#alu] = "not(" & stage3Regs[#X3] & ")"
      "neg":
        logic[#alu] = -1 * stage3Regs[#X3]
      "and", "andi", "or", "ori", "shr", "shra", "shl", "shc":

        logic[#alu] = stage3Regs[#X3] && IR[3][#op] && stage3Regs[#Y4]
    end case
  end if
end


on getDecode4

  global controlSignals, stage4Regs, logic

  if controlSignals[4][#load] then
    logic[#decode4] = "getDataMem from M[Z4]"
  else if controlSignals[4][#ladr] or controlSignals[4][#branch] or¬
 controlSignals[4][#alu] then
    logic[#decode4] = stage4Regs[#Z4]
  else if controlSignals[4][#store]= true then
    logic[#decode4] = stage4Regs[#MD4]
  end if
```

```
end


on getDecode5
  global controlSignals, stage5Regs, logic

  if controlSignals[5][#regWrite] then
    logic[#decode5] = "loadDataMem from Z5 to R[ra]"
  end if
end



on memInit
  --initializes a global vector called mainMem whose content will
consist of
  --the contents of main memory at any given instant. After the vector
is
  --initialized, calls method "loadMainMemFields" to load the fields so
the
  --contents can be viewed by user

  global MAINMEM, NumInst
  NumInst = 0

  --put initial values in general reg
  MAINMEM = [m4000:1, m4004:5, m4008:4000, m4012:300, m4016:40, m4020:-
12, ¬
m4024:8, m4028:-4, m4032:4024]
  loadMainMemFields

end

on updateMainMem mainMemChangeList
  --Takes a property list that has common properties of the global
vector
  --MAINMEM and inputs the value into MAINMEM
  --
  --arguments: mainMemChangeList - a property vector containing common
properties
  --                                 of the global vector MAINMEM

  global MAINMEM

  --put changes into mainMemList
  numChanges = count("mainMemChangeList")
  repeat with i = 1 to numChanges
    memAddres = getPropAt("mainMemChangeList", i)
    MAINMEM[#memAddres] = mainMemChangeList[i]
  end repeat
  loadMainMemFields
end




--loads mainMem fields to be loaded
on loadMainMemFields
  global MAINMEM
  memFields = ["4000","4004", "4008", "4012", "4016", "4020", "4024",
"4028",¬
  "4032"]

  repeat with i= 1 to count(MAINMEM)
    x = memFields[i]
    put MAINMEM[i]
    set member(x).text = string(MAINMEM[i])
  end repeat
end

--put initial values in mem space
on genRegInit
  global GENREG
  --put initial values in mem space
  GENREG = [R0:0, R1:4000, R2:1, R3:4036, R4:12, R5:6, R6:4, R7:empty,
R8:empty, R9:-16]

  --put values in  genReg fields
  loadGenRegFields
end

on loadGenRegFile
  global GENREG, IR, regFile, switch, Z5, logic
  case IR[2][#instType] of
    1, 2, 3, 5, 6, 7, 77:
      regFile[#a3] = IR[5][#ra]
      regIndex = chars(regFile[#a3], 2, 2)

      if logic[#decode5] = true then
        GENREG[regIndex] = Z5
      end if
    1, 4, 5, 6, 7, 77:
      regFile[#a1] = IR[2][#rb]
      regIndex = integer(regFile[#a1].char[2]) + 1
      regFile[#R1] = GENREG[regIndex]

      regFile[#a2] = switch[#Mp2]
      regIndex = integer(regFile[#a2].char[2]) + 1
      regFile[#R2] = GENREG[regIndex]

  end case

  loadGenRegFields
end


--loads genReg fields to be loaded
on loadGenRegFields
  global GENREG

  genRegFields = ["R0","R1", "R2", "R3", "R4", "R5", "R6", "R7", "R8",
"R9"]

  repeat with i= 1 to count(GENREG)
    set member(genRegFields[i]).text = string(GENREG[i])
  end repeat
end
```

```
on getInstFields instFormat
  global InstMemList, NumInst

  Op = member("Op").text
  ra = sendAllSprites(#DropList_Selection, "raList")
  rb = sendAllSprites(#DropList_Selection, "rbList")
  rc = sendAllSprites(#DropList_Selection, "rcList")
  c1 = member("c1").text
  c2 = member("c2").text
  cond = sendAllSprites(#DropList_Selection, "cond")
  shiftCount =  member("count").text


  case (instFormat) of
    1:
      instruction = [instType:1, instAddress:"", op:Op, ra:ra[#text],¬
 rb:rb[#text], c2:c2]
    2:
      instruction = [instType:2, instAddress:"", op:Op, ra:ra[#text],¬
 c1:c1]
    3:
      instruction = [instType:3, instAddress:"", op:Op, ra:ra[#text],
rc:rc[#text]]
    4:
      instruction = [instType:4, instAddress:"", op:Op, rb:rb[#text],
rc:rc[#text], cond:cond[#text]]
    5:
      instruction = [instType:5, instAddress:"", op:Op, ra:ra[#text],
rb:rb[#text], ¬
rc:rc[#text], cond:cond[#text]]
    7:
      instruction = [instType:instFormat, instAddress:"", op:Op,
ra:ra[#text], rb:rb[#text],¬
shiftCount:shiftCount]
    6, 77:
      instruction = [instType:instFormat, instAddress:"", op:Op,
ra:ra[#text], rb:rb[#text], rc:rc[#text]]
    8:
      instruction = [instType:instFormat, instAddress:"", op:Op]

  end case


  -- verfiy correct field inputs
  error = instFieldVerify (instruction)

  if (error[#error] = false) then

    --update hazard list
    hazardCheck(instruction)

    -- insert inst fields into inst mem
    --    add(InstMemList, instruction)
    NumInst = NumInst + 1
    loadInstMem instruction

    -- clear fields
    set member("Op").text = empty
    set member("c2").text = ""
    set member("c1").text = ""
    set member("count").text = ""

    -- go to select another inst
    go to frame "chooseInst"
  else
    alert("ERROR" && error[#errorType])
  end if
end


-- verifies inst field inputs from user; if incorrect alert msg
on instFieldVerify instruction

  instEntryError = [error:false, errorType:""]

  case(instruction[#instType]) of
    1:
      if (instruction[#ra] = instruction[#rb]) then
        instEntryError = [error:true, errorType:"matchingArguments"]
        put instEntryError


      else if (instruction[#c2] < -100000) or (instruction[#c2] >
100000)  then
        instEntryError = [error:true, errorType:"c2OutOfBounds"]
        put instEntryError
        member("c2").text = empty
      end if

    2:
      if (instruction[#c1] < -100000) or (instruction[#c1] > 100000)
then
        instEntryError = [error:true, errorType:"c1OutOfBounds"]
        put instEntryError
        member("c1").text = empty
      else if instruction[#rb] = "rb=0" then
        instEntryError = [error:true, errorType:"rb=0 error"]
        put instEntryError

      end if--else return

    4:
      if (instruction[#rb] = instruction[#rc]) then
        instEntryError = [error:true, errorType:"matchingArguments"]
        put instEntryError
      else if instruction[#rb] = "rb=0" then
        instEntryError = [error:true, errorType:"rb=0 error"]
        put instEntryError


      end if

    5, 6, 77:
      if ((instruction[#rb] = instruction[#ra]) or (instruction[#rb] =
instruction[#rc]) or (instruction[#ra] = instruction[#rc])) then
        instEntryError = [error:true, errorType:"matchingArguments"]
        put instEntryError
      else if instruction[#rb] = "rb=0" then
        instEntryError = [error:true, errorType:"rb=0 error"]
        put instEntryError
      end if

    3:
      if (instruction[#ra] = instruction[#rc]) then
        instEntryError = [error:true, errorType:"matchingArguments"]
        put instEntryError
      else if instruction[#rb] = "rb=0" then
        instEntryError = [error:true, errorType:"rb=0 error"]
        put instEntryError
      end if


    7:
      if (instruction[#ra] = instruction[#rb]) then
        instEntryError = [error:true, errorType:"matchingArguments"]
        put instEntryError

      else if (instruction[#shiftCount] <= 0) or
(instruction[#shiftcount] >= 32)  then
        instEntryError = [error:true, errorType:"countOutOfBounds"]
        put instEntryError
      else if instruction[#rb] = "rb=0" then
        instEntryError = [error:true, errorType:"rb=0 error"]
        put instEntryError
      end if


  end case


  return instEntryError
end

on hazardCheck (instruction)
  global rb_rc_hazard, ra_hazard
  rb_rc_hazard = ["", "", "", ""]
  ra_hazard = ["", "", "", ""]

  repeat with i = 1 to 3
    rb_rc_hazard[i + 1] = rb_rc_hazard[i]
    ra_hazard[i+1] = ra_hazard[i]
  end repeat
  i = 1

  case (instruction[#instType]) of
    1:
      rb_rc_hazard[i] = instruction[#ra]

      if instruction[#rb] = "rb=0" then
        ra_hazard[i] = ""
      else
        ra_hazard[i] = instruction[#rb]
      end if
    2:
      rb_rc_hazard[i] = instruction[#ra]
    3:
      rb_rc_hazard[i] = instruction[#ra]
      ra_hazard[i] = instruction[#rc]
    4:
        ra_hazard[i] = instruction[#rb] && instruction[#rc]

      5, 6, 77:
        rb_rc_hazard[i] = instruction[#ra]
        ra_hazard[i] = instruction[#rb] && instruction[#rc]
      7:
        rb_rc_hazard[i] = instruction[#ra]
        ra_hazard[i] = instruction[#rb]
      8:
        rb_rc_hazard[i] = ""
        ra_hazard[i] = ""
    end case

    --write data to field
    member("rb_rc_Hazards").text = rb_rc_hazard[1] && rb_rc_hazard[2] && ¬
  rb_rc_hazard[3] && rb_rc_hazard[4]
    member("raHazards").text = ra_hazard[1] && ra_hazard[2] &&
  ra_hazard[3] && ¬
  ra_hazard[4]
    return
  end

  --Load inst into mem. Assign location numbers.
  on loadInstMem instruction
    global NumInst
    global InstMemList

    --input address
    address = 1996 + (4 * NumInst)
    --address = string(address)
    instruction[#instAddress] = address

    --load into mem
    setAt(InstMemList, NumInst, instruction)
    if NumInst > 9 then
      return
    else
      --load inst fields
      loadInstFields instruction
    end if
    return
  end


  on load15Inst
    global NumInst, InstMemList
    numNOP = 15 - NumInst
    x = NumInst
    --

    nopInst = [instType:8, instAddress:"", op:"nop"]

    repeat with i = 1 to numNOP
      x = x + 1
      add(InstMemList, nopInst)

    end repeat
    load15NopAddress
    return
  end
```

```
on load15NopAddress
  global InstMemList, NumInst
  startIndex = NumInst + 1
  numNOP = 15 - NumInst
  repeat with i = startIndex to 15
    InstMemList[i][#instAddress] = 1996 + (4*i)
  end repeat
end




on loadInstFields instruction
  --load inst mem fields with values

  case (instruction[#instType]) of
    1:
      if ((instruction[#op] = "ld") or (instruction[#op] = "st") or ¬
(instruction[#op] = "la")) then
        if instruction[#rb] = "rb=0" then
          set member(string(instruction[#instAddress])).text =
instruction[#op] && ¬
instruction[#ra] && instruction[#c2]
        else
          set member(string(instruction[#instAddress])).text =
instruction[#op] && ¬
instruction[#ra] && instruction[#c2] & "(" & instruction[#rb] & ")"
        end if

      else
        set member(string(instruction[#instAddress])).text =
instruction[#op] && ¬
 instruction[#ra] && instruction[#rb] && instruction[#c2]
      end if

    2:set member(string(instruction[#instAddress])).text =
instruction[#op] && ¬
instruction[#ra] && instruction[#c1]
    3:set member(string(instruction[#instAddress])).text =
instruction[#op] && ¬
instruction[#ra] && instruction[#rc]
    4:set member(string(instruction[#instAddress])).text =
instruction[#op] && instruction[#rb] ¬
 && instruction[#rc] && instruction[#cond]
    5:
      set member(string(instruction[#instAddress])).text =
instruction[#op] && instruction[#ra] ¬
 && instruction[#rb] && instruction[#rc] && instruction[#cond]
    6, 77:
      set member(string(instruction[#instAddress])).text =
instruction[#op] && instruction[#ra] ¬
&& instruction[#rb] && instruction[#rc]
    7:
      set member(string(instruction[#instAddress])).text =
instruction[#op] && instruction[#ra] ¬
&& instruction[#rb] && instruction[#shiftCount]
    8:
      set member(string(instruction[#instAddress])).text =
instruction[#op]
  end case
  return

end


on initInstFetch
  initInstFormat
  clearInstMem
end


on initInstFormat
  global InstFormatList

  --set global list for instruction format

  set instFormat1 = ["ld", "st", "la", "addi", "andi", "ori"]
  set instFormat2 = ["ldr", "str", "lar"]
  set instFormat3 = ["neg", "not"]
  set instFormat4 = ["br"]
  set instFormat5 = ["brl"]
  set instFormat6 = ["add", "sub", "and", "or"]
  set instFormat7 = ["shr", "shra", "shl", "shc"]
  set instFormat8 = ["nop", "stop"]

  set InstFormatList = [instFormat1, instFormat2, instFormat3,
instFormat4, instFormat5, instFormat6, instFormat7, instFormat8]

  --set misc instruction format variables
  set countField = false
  set condCodeField = false

end

--gets users input from field list and determines inst type
on getUserInstChoice opCodeSelected
  opCode = opCodeSelected
  put opCode into field "Op"

  instFormat = getInstFormat (opCodeSelected)
  put instFormat
  getFields instFormat

end getUserInstChoice



--gets instruction format depending on opcode of the instruction
on getInstFormat (opCode)
  global InstFormatList
  set instFormat = 0
  i = 1
  --search through instruction array for instruction match
  instFound = FALSE
  repeat while instFound = false --search through each row of the array

    repeat with j = 1 to count(InstFormatList[i])
      if opCode = getAt(InstFormatList[i], j) then
        instFound = true
        instFormat = i
      end if
    end repeat
    i = i + 1
  end repeat
  return instFormat --return instruction format
end

-- gets fields for instructions depending on instruction format
on getFields instFormat
  case (instFormat) of
    1:go to frame "instFormat1"
    2:go to frame "instFormat2"
    3:go to frame "instFormat3"
    4:go to frame "instFormat4"
    5:go to frame "instFormat5"
    6:go to frame "instFormat6"
    7:go to frame "instFormat7"
    8:go to frame "instFormat8"
  end case
end

-- DESCRIPTION --

on getBehaviorDescription me
  return "¬
DROPDOWN LIST"&RETURN&RETURN&"¬
Drop this behavior on a Field member to create a pop-up list.  ¬
Animations continue while the the list is kept open."&RETURN&RETURN&"¬
When the user clicks on the sprite, the dropdown list opens to reveal
all ¬
its items.  if the user immediately releases the mouse, the menu remains
¬
open until the next click.  When the user selects a menu item, the menu
¬
closes up to displays the selected item.  If the user clicks elsewhere,
¬
the menu closes to display the previously selected
item."&RETURN&RETURN&"¬
You can use one of two modes for the Dropdown list:"&RETURN&"¬
1) To allow the user to make a selection"&RETURN&"¬
2) To execute a simple command."&RETURN&RETURN&"¬
SELECTION"&RETURN&"¬
In the first case, you will need to determine what the user selected.  ¬
To interrogate the Dropdown list, use syntax similar to the
following:"&¬
RETURN&RETURN&"¬
   put sendAllSprites (#DropList_Selection,
'listName')"&RETURN&RETURN&"¬
This returns a property list with all the necessary information:"&¬
RETURN&RETURN&"¬
-- [#item: 1, #text: 'First choice', #type: #content, #sprite: 1]"&¬
RETURN&RETURN&"¬
See the 'Notes for developers' in the script itself for more details."&¬
RETURN&RETURN&"¬
You can choose any character to act as a checkmark to indicate the
previous ¬
selection when the dropdown list is open.  Depending on the font you
use, you ¬
may wish to use a checkmark followed by a space.  Reopen the Behavior ¬
Parameters dialog to make such a change."&RETURN&RETURN&"¬
EXECUTION"&RETURN&"¬
```

```
You can choose to execute three types of command:"&RETURN&"¬
a) go marker (<selected item>)"&RETURN&"¬
b) go movie '<selected item>'"&RETURN&"¬
c) do '<selected item>'"&RETURN&RETURN&"¬
The type of command depends on the contents of the list.  The behavior
can ¬
automatically create a list of markers in the current movie, or movies
in the ¬
current folder... or it can leave the contents of the Field as they are.
In ¬
this last case, choosing 'Execute' makes the behavior treat the selected
item ¬
as a Lingo command.  You should include handlers in a movie script to
deal ¬
with such commands."&RETURN&RETURN&"¬
TIP: Place the dropdown list sprite in a high channel where it will not
be covered ¬
by any other sprites."&RETURN&RETURN&"¬
PERMITTED MEMBER TYPES:"&RETURN&"Field members"&RETURN&RETURN&"¬
PARAMETERS:"&RETURN&"¬
* Name of the list (used in sendAllSprite calls)"&RETURN&"¬
* Purpose - Choose between:"&RETURN&"¬
  - Marker: creates a list of markers in current movie"&RETURN&"¬
  - Movie: creates a list of movies with the same pathName"&RETURN&"¬
  - Field contents: uses the current contents of the field"&RETURN&"¬
* Action on selection - Choose between:"&RETURN&"¬
  - Execute: go movie | go marker | do selectedLine"&RETURN&"¬
  - Select:  return the selected item if called to do so"&RETURN&"¬
* Checkmark to indicate currently selected item"&RETURN&"¬
* Standard style: deselect this option if you want to give the Field
member a ¬
particular border, margin or shadow."&RETURN&RETURN&"¬
PUBLIC METHODS:"&RETURN&"¬
=> Get info on currently selected item"&RETURN&"¬
=> Set the contents of the dropdown list"&RETURN&"¬
=> Toggle between Execute and Select modes"&RETURN&"¬
=> Get behavior reference"
end getBehaviorDescription


on getBehaviorTooltip me
  return "¬
Use with Field members only."&RETURN&RETURN&"¬
Turn a Field into a pop-up list to execute commands"&RETURN&"¬
or store selected data.  See the Behavior Description"&RETURN&"¬
for tips on executing items with the 'do' command or"&RETURN&"¬
accessing the currently selected item using Lingo."&RETURN&RETURN&"¬
Options: create a list of movies with the same path"&RETURN&"¬
name, or a list of markers in the current movie."
end getBehaviorTooltip



-- NOTES FOR DEVELOPERS --
--
-- The handlers of this behavior fall into three types: those that
determine
-- the contents of the list, those that react to the user's mouse
clicks, and
-- those that allow Lingo access to the current selection.
--
```

```
-- CREATING THE CONTENTS
-- The author has a choice of three types of content: #movie, #marker or
-- the current contents of the field.  The latter two are easy to deal
with
-- since the information already exists in the form of string.  (The
labelList
-- is in fact a string, not a list).
--
-- Creating a RETURN delimited string of movies is more complex.  We
want to
-- list all movies in the same folder as the current movie, or which are
in
-- subfolders of this folder.  The CreateMovieLists handler uses
recursion.
-- The getNthFileNameInFolder function is used to look at each item in
the
-- folder in turn.  If it is a movie, it is added to the list.  If not,
it is
-- treated as if it were itself a folder.  The MovieList handler calls a
clone
-- of itself to examine the contents of this new folder.  Any new movies
found
-- are added to the growing movies variable.

-- Once the content has been established, the dimensions of the list can
-- be calculated.  Setting the rect of a text member changes the size of
the
-- onstage sprite.  It does not in fact change the rect of the member.
This
-- will always return the full height of the text.  This behavior
exploits this
-- fact to create the myOpenRect property.  A RETURN character is then
added to
-- the end of the list, to ensure that the hilite of the last line of
the list
-- runs the full width of the field.

-- ALLOWING FOR CHECKMARKS
-- Actually, a RETURN delimited string is not quite enough.  I place 3
spaces
-- at the beginning of each line, to make room for a checkmark.  if the
menu is
-- set to "select" (rather than ("execute"), then it needs to display a
-- checkmark to remind the user which item was selected previously.
Most
-- characters that would be used as a checkmark, in most fonts, will be
about
-- two spaces wide.  So I place the checkmark in spaces one and two.  If
you
-- use a proportional font, this will make the item jump one space to
the left.
-- To solve this use a character followed by a space as your checkmark:
"* ".
--
-- Rather than add and subtract spaces constantly, I chose to create a
list
-- with number items, containing the original text: myItemsList.  This
has the
-- additional advantage of storing the frame numbers for each marker, if
-- you choose to create a list of markers.  I am indebted to Pim van
Bochoven
```

```
-- for showing me how to create a list which includes duplicate marker
names.

-- INTERACTING WITH THE USER
-- The major difficulty here is to determine the size and position of
the open
-- list.  Ideally, the list should open so that the current selection is
under
-- the cursor.  If the list is too near the top or bottom of the Stage
for this
-- to happen, its position is shifted up or down the appropriate number
of
-- lines.
--
-- This is not the standard behavior of a pop-up menu.  Standard menu
behavior
-- would require either an Xtra or more than one sprite.
--
-- The difference between this list behavior and a pop-up menu becomes
even
-- clearer if the list is too long to fit on the Stage.  If this
happens, a
-- scrollbar is provided.
--
-- A mouseUp can have two meanings: a fast click will hold the list
open, a
-- longer click indicates that the user has made a selection.  the value
of
-- the ticks when the mouse is first pressed is saved in myClickTicks.
This
-- value is compared against the ticks as soon as the mouse is released.
If
-- the difference is less than 30 ticks (half a second) the list wil
remain
-- open.  Otherwise, the behavior considers that a selection has been
made.
--
-- If the list remains open, a click elsewhere should close it.  The
-- prepareFrame handler checks if the clickOn is the number of the list
-- sprite.  If not, the clicks was elsewhere.
--
-- When the list closes, two things must happen: the rect of the member
has
-- to be set to the height of a single line, and the contents of the
field must
-- be scrolled so that the selected item appears in the remaining space.
The
-- scrollTop property gives the height in pixels between the start of
the text
-- and the top of the visible line of text.

-- Note the simplicity of the ScrollTo handler.

-- ACCESSING THE CURRENT SELECTION
-- The DropList_Selection handler gives flexible access to the current
-- selection.  If only one dropdown list appears in the current frame,
use..
--
--   put sendAllSprites (#DropList_Selection)
--
```

```
-- ...to print out a list in the Message Window.  If more than one list
is
-- present, then you can either include the list's name in your call:
--
--   put sendAllSprites (#DropList_Selection, "Markers")
--   -- [#item: 1, #text: "Intro, #type: #marker, #sprite: 1]
--
-- ... or send in an empty property list for each dropdown list to fill
in with
-- its own details:
--
--   put sendAllSprites (#DropList_Selection, [:])
--   -- ["Markers": [#item: 1, #text: "Intro", #type: #marker, #sprite:
1],
--   "Movies": [#item: 5, #text: "Main", #type: #movie, #sprite: 2]]
--
-- You could also target the list's sprite directly, with a sendSprite
call.
-- Any of these calls will return a list specifying:
-- * the number of the selected item
-- * the text of the selected item
-- * the type of data concerned (#movie | #marker | #content)
-- * the dropdown list's spriteNum
--
-- If you collect data for all dropdown lists in a property list, then
the name
-- of each dropdown list will also be returned (as the property for each
entry).

-- EXECUTING CUSTOM LINGO
-- In the Behavior Parameters dialog you can set the list to use the
-- "Current contents of the field" and to "Execute... do selectedLine".
The
-- behavior will now treat each item in the dropdown list as a Lingo
command.
-- You will need to provide handlers in a movie script to treat these
commands.
--
-- For example, your dropdown list could contain the following items:
--
--    Register
--    Help
--    Quit
--
-- Only "Quit" is a Lingo keyword that could be executed directly.  For
the
-- other two items you would need a movie script with the following
handlers:
--
--   on Register
--     go marker ("Register")
--   end
--
--   on Help
--     open window "Help"
--   end

-- ALTERING THE CONTENTS OF THE DROPDOWN LIST
-- You can use a #DropList_SetContents call to change the contents of
the
-- dropdown list on the fly.
-- If you send either a RETURN delimited string or a list as a
parameter, the
-- dropdown list will now behave as if you had chosen  "Current contents
of the
-- field" in the Behavior Parameters dialog.  The default item will be
the
-- first item in your list.  Example:
--
--  sendSprite (1, #DropList_SetContents, ["Register","Help", "Quit"])
--
-- If the dropdown list is set to "Execute... do selectedLine", then
make sure
-- that there are handlers in a movie script to deal with each command.
--
-- You can also use the symbols #marker and #movie as a parameter.  This
will
-- automatically create a list of either markers in the current movie,
or
-- movies in the current folder and subfolders.  Examples:
--
--   sendSprite (1, #DropList_SetContents, #marker)
--   sendAllSprites (#DropList_SetContents, #movie, "List 1")

-- CHOOSING TO SELECT OR TO EXECUTE AN ITEM
-- You can toggle between the selection and execution modes using a
-- #DropList_ToggleExecution call.  This takes one or two parameters:
the new
-- mode and/or the name of the list to toggle.  If you use both
parameters, the
-- new mode should appear first.  Example:
--
--    sendAllSprites (#DropList_ToggleExecution, #execute, "Movies")
--
-- If you do not specify a mode, the mode will switch between #execute
and
-- #select.  If you do not specify a list, then all dropdown lists in
the
-- current frame will be affected.



-- HISTORY --

-- 12 - 13 September 1998: written for the D7 Behaviors Palette by James
Newton
-- 28- 30 October 1998:  rewritten with help and guidance from Pim van
Bochoven
-- to include a selection checkmark and to allow the use of markers with
-- duplicate names.
-- 13 November 1998: additional Public Method handlers added and
commented.


-- PROPERTIES --

property spriteNum
property mySprite
-- author-defined parameters
property myName
property myContent
property myAction
property myCheckmark
property myStandard
-- internal properties
property myListMember
property myField
property myItemsList
property myRestoreString
property myDisplayString
-- dimensions
property myItemHeight
property myOpenRect
property myClosedRect
property myClosedLoc
property myOpenHeight
property myStageHeight
property myStageWidth
-- selection
property mySelectedItem
property myListIsOpen
property myClickTicks
property myLastHilite



-- EVENT HANDLERS --

on beginSprite me
  Initialize me
end beginSprite


on mouseDown me
  if not myListIsOpen then OpenList me
end mouseDown


on prepareFrame me
  CheckListState me
end prepareFrame


on mouseUp me
  CheckClick me
end mouseUp


on mouseUpOutside me
  CloseList me
end mouseUpOutside



-- CUSTOM HANDLERS: UTILIZATION --

on CheckListState me
  if myListIsOpen then
    if the clickOn <> spriteNum then
      -- Click outside list while it is held open
      CloseList me
    else
      HiliteSelection me
    end if
  else
    if myContent = #marker and myAction then
      markerNumber      = GetCurrentMarker (me)
      if mySelectedItem = markerNumber then exit

      mySelectedItem    = markerNumber
      ScrollTo me, mySelectedItem
    end if
  end if
end CheckListState


on OpenList me -- sent by mouseDown
  myClickTicks = the ticks
  myListIsOpen = TRUE
  myDisplayString = myRestoreString
  if not myAction then
    put myCheckMark into myDisplayString.line[mySelectedItem].char
[1..2]
  end if
  myListMember.text = myDisplayString

  currentScroll = myListMember.scrollTop

  if myOpenHeight <= myStageHeight then
    -- Show whole list to best advantage
    overShoot = currentScroll - myClosedLoc[2]
    if overShoot < 0 then
      overShoot = myOpenHeight - overShoot - myStageHeight
      if overShoot < 0 then
        -- Simplest case
        mySprite.locV = myClosedLoc[2] - currentScroll

      else
        -- List too low: shift it up so it will appear in full
        lineAdjust = ((overShoot - 1) / myItemHeight) + 1
        pixelAdjust = (lineAdjust * myItemHeight) - overShoot
        openTop = myStageHeight  - myOpenHeight - pixeladjust
        mySprite.locV = openTop
      end if

    else
      -- List too high: shift it down so it will appear in full
      lineAdjust = ((overShoot - 1) / myItemHeight) + 1
      pixelAdjust = lineAdjust * myItemHeight
      openTop = pixelAdjust - overShoot
      mySprite.locV = openTop
    end if
    myListMember.scrollTop = 0
    myListMember.rect = myOpenRect

  else
    -- List is too long to show in full:  show as much as possible...
    mySprite.locV = -2
    clippedRect = myOpenRect.duplicate()
    clippedRect[4] = myStageHeight
    myListMember.rect = clippedRect
    --...and let it a scroll
```

```
      myListMember.boxType = #scroll
      if mySprite.right > myStageWidth then
        spriteWIdth = mySprite.right - mySprite.left
        mySprite.locH = myStageWidth - spriteWIdth
      end if
      scrollAdjust = myClosedLoc[2] - mySprite.locV
      myListMember.scrolltop = currentScroll - scrollAdjust
    end if

    updateStage
  end OpenList


  on HiliteSelection me  -- sent by prepareFrame
    if the mouseMember <> myListMember then
      if myLastHilite then
        -- Clear hilite to indicate that no action will be taken
        myLastHilite = 0
        hilite char the maxInteger of field myField
      end if
      exit

    else
      if myListMember.boxtype = #scroll then AutoScroll me
      -- Hilite the line under the mouse
      listLocV = mouseV() - mySprite.locV + myListMember.scrollTop
      mouseItem = (listLocV / myItemHeight) + 1
    end if

    if mouseItem = myLastHilite then exit
    if mouseItem > myItemsList.count then
      myLastHilite = 0
      hilite char the maxInteger of field myField
      exit
    end if

    myLastHilite = mouseItem
    if mouseItem = 1 then
      firstCharToHilite = 1
    else
      textBeforeMouseItem = line 1 to (mouseItem - 1) of myDisplayString
      firstCharToHilite = the number of chars of textBeforeMouseItem + 2
      -- Two extra characters = invisible RETURN + first char of mouseItem
    end if
    mouseItemLength = the number of chars of line mouseItem of
  myDisplayString
    lastCharToHilite = firstCharToHilite + mouseItemLength
    hilite char firstCharToHilite to lastCharToHilite of field myField
  end HiliteSelection


  on AutoScroll me -- sent by HiliteSelection
    scrollDownHeight = myItemHeight / 2
    scrollUpHeight = myStageHeight - myItemHeight / 2
    currentScroll = myListMember.scrollTop
    if mouseV() < scrollDownHeight then
      if currentScroll <> 0 then
        newScroll = currentScroll - scrollDownHeight
        myListMember.scrollTop = max (0, newScroll)
      end if
    else if mouseV() > scrollUpHeight then
    maxScroll = myOpenHeight - myStageHeight
    if currentScroll <> maxScroll then
      newScroll = currentScroll + scrollDownHeight
      myListMember.scrollTop = min (maxScroll, newScroll)
    end if

  end if
end AutoScroll


on CheckClick me -- sent by mouseUp
  if the ticks - myClickTicks < 30 then
    -- Fast click: hold list open and get ready to react to the next
click
    myClickTicks = 0
  else
    if myLasthilite then
      -- Treat selection and close
      mySelectedItem = myLastHilite
    end if
    CloseList me
    if myAction and myLastHilite then
      Execute me
    end if
  end if
  myLastHilite = 0
end CheckClick


on ScrollTo me, theLine -- sent by Initialize, CloseList
  myListMember.scrollTop = -10
  -- D7 won't set the scrollTop if it thinks it hasn't changed
  myListMember.scrollTop = myItemHeight * (theLine - 1)
end ScrollTo


on CloseList me -- sent by mouseUpOutside, HiliteSelection, CheckClick
  hilite char the maxInteger of field myField
  if not myAction then
    myListMember.text = myRestoreString
  end if
  mySprite.loc = myClosedLoc
  myListMember.boxType = #fixed
  myListMember.rect = myClosedRect
  ScrollTo (me, mySelectedItem)
  myListIsOpen = FALSE
  if myContent <> #marker then updateStage
end CloseList


on Execute me -- sent by CheckClick
  theItem = myItemsList[mySelectedItem]
  case myContent of
    #movie:
      if not (the movieName starts (theItem&".")) then
        go movie theItem
      end if
    #marker:  go myItemsList.getPropAt(mySelectedItem)
    #content: do theItem
  end case
end Execute
```

```
-- CUSTOM HANDLERS: INITIALIZATION --

on Initialize me -- sent by beginSprite
  mySprite = sprite(me.spriteNum)
  myListMember = mySprite.member

  -- Error checking
  memberType = myListMember.type
  if memberType <> #field then
    ErrorAlert (me, #invalidMemberType, memberType)
  end if
  -- End of error checking

  myField = myListMember.number
  -- Convert to symbols
  case myContent of
    "Current contents of the field":  myContent = #content
    "Markers in this movie":          myContent = #marker
    "Movies with the same path name": myContent = #movie
  end case

  -- Convert to boolean
  case myAction of
    "Select:  return the selected item when called":
      myAction = FALSE
    "Execute: go movie | go marker | do selectedLine":
      myAction = TRUE
  end case

  -- Ensure that the field properties are properly set
  myListMember.wordWrap  = FALSE
  myListMember.alignment = "left"
  myListMember.boxType = #fixed
  if myStandard then
    myListMember.border = 1
    myListMember.margin = 2
    myListMember.boxDropShadow = 2
  end if

  -- Determine the field's content
  CreateItems me
  mySelectedItem = DefaultItem (me)
  SetDimensions me

  -- Display closed list
  myListMember.rect = myClosedRect
  ScrollTo (me, mySelectedItem)
end Initialize



on CreateItems me -- sent by Initialize
  case myContent of
    #content:
      CreateContentsLists (me)
    #marker:
      myRestoreString = AddSpaces (me, the labelList)
      myItemsList     = GetMarkedFrames (me)
    #movie:
      myRestoreString = ""
      myItemsList     = [:]
      saveDelimiter = the itemDelimiter
      the itemDelimiter = "."
      CreateMovieLists (me, the moviePath)
      set the itemDelimiter = saveDelimiter
  end case
end CreateItems


on AddSpaces me, theText
  -- Adds three spaces to the beginning of every line
  repeat while the last char of theText = RETURN
    delete the last char of theText
  end repeat

  newString = EMPTY
  lineCount = theText.line.count
  repeat while lineCount
    theItem = theText.line[lineCount]
    put "   "&theItem&RETURN before newString
    lineCount = lineCount - 1
  end repeat
  return newString
end AddSpaces


on CreateContentsLists me
  theText = myListMember.text
  repeat while the last char of theText = RETURN
    delete the last char of theText
  end repeat

  myRestoreString = EMPTY
  myItemsList     = [:]
  lineCount       = theText.line.count
  repeat with i = 1 to lineCount
    theItem = theText.line[i]
    -- Strip myCheckmark and any spaces
    if SPACE&myCheckmark contains theItem.char[1] then
      delete theItem.char[1]
      repeat while theItem.char[1] = " "
        delete theItem.char[1]
      end repeat
    end if
    -- Ensure that the line starts with 3 spaces
    myRestoreString = myRestoreString&"   "&theItem&RETURN
    myItemsList.addProp(i, theItem)
  end repeat
end CreateContentsLists


on GetMarkedFrames me
  markerList = [:]
  sort markerList

  lastCheckedMarker = 0
  if marker (1) <> marker (-the maxInteger / 2) then
    -- We're after the first marker
    repeat with i = 0 down to -the maxInteger
```

```
      checkMarker = marker(i)
      if checkMarker = lastCheckedMarker then exit repeat
      lastCheckedMarker = checkMarker
      markerList.addProp (checkMarker, 0)
    end repeat
  end if

  if marker (0) <> marker (the maxInteger / 2) then
    -- We're before the last marker
    repeat with i = 1 to the maxInteger
      checkMarker = marker(i)
      if checkMarker = lastCheckedMarker then exit repeat
      lastCheckedMarker = checkMarker
      markerList.addProp(checkMarker, 0)
    end repeat
  end if

  i = markerList.count()
  theLabels = the labelList
  repeat while i
    markerList [i] = theLabels.line [i]
    i = i - 1
  end repeat
  return markerList
end GetMarkedFrames


on CreateMovieLists me, folderName -- sent by GetListItems
  -- Recursive handler
  if (the machineType = 256) then
    fileDelimiter = "\"
  else
    fileDelimiter = ":"
  end if

  fileCount = 0
  repeat while TRUE
    fileCount = fileCount + 1
    theFileName   = getNthFileNameInFolder (folderName, fileCount)
    if theFileName = EMPTY then
      -- No more files
      return -- movieString
    else
      -- Check if theFile is a movie
      case item 2 of theFileName of
        "dir", "dxr", "dcr":
          theMovie        = item 1 of theFileName
          myRestoreString = myRestoreString&"    "&theMovie&RETURN
          movieCount      = myItemsList.count() + 1
          myItemsList.addProp(movieCount, theMovie)
        otherwise
          -- Check if theFile is not in fact a folder: use recursion
          CreateMovieLists (me, folderName&theFileName&fileDelimiter)
      end case
    end if
  end repeat
end MovieList


on DefaultItem me -- sent by Initialize
  case myContent of
```

```
    #content: return 1
    #marker:  return GetCurrentMarker (me)
    #movie:
      saveDelimiter = the itemDelimiter
      set the itemDelimiter to "."
      shortName = item 1 of the movieName
      set the itemDelimiter = saveDelimiter
      return myItemsList.getPos (shortName)
  end case
end DefaultItem


on SetDimensions me
  -- Determine the dimensions of the list
  saveLastChar      = the last char of myRestoreString -- Should be
RETURN
  delete the last char of myRestoreString
  myListMember.text = myRestoreString
  myItemHeight      = myListMember.lineHeight
  myOpenRect        = myListMember.rect
  myClosedRect      = myOpenRect.duplicate()
  myClosedRect[4]   = myItemHeight + (myListMember.margin / 2)
  myClosedLoc       = mySprite.loc
  addedHeight       = (myListMember.margin * 2) +
myListMember.boxDropShadow
  myOpenHeight      = myOpenRect.bottom + addedHeight

  windowRect        = (the activeWindow).rect
  myStageHeight     = windowRect.bottom - windowRect.top
  myStageWidth      = windowRect.right  - windowRect.left
  myRestoreString   = myRestoreString&saveLastChar
  myListMember.text = myRestoreString
end SetDimensions


on GetCurrentMarker me -- sent by Initialize
  -- Also used to update display if playback head moved by some other
means
  markerPosition = myItemsList.findPos(the frame)
  if not markerPosition then
    markerPosition = myItemsList.findPosNear(the frame) - 1
  end if
  return max (1, markerPosition)
end GetCurrentMarker



-- PUBLIC METHOD (response to #sendSprite, #sendAllSprites, #call) --

on DropList_Selection me, propListOrString
  -- Returns the current selection of the dropdown list.  If you have
several
  -- dropdown lists with different names, then you can use
sendAllSprites call
  -- with a property list as a parameter.  Example:
  --
  --   put sendAllSprites (#DropList_Selection, [:])
  --   -- ["Markers": [#item: 1, #text: "Intro", #type: #marker,
#sprite: 1],
  --   "Movies": [#item: 5, #text: "Main", #type: #movie, #sprite: 2]]
  -- If you want the current selection of a particular list then use its
name:
  --
  --   put sendAllSprites (#DropList_Selection, "Markers")
  --   -- [#item: 1, #text: "Intro", #type: #marker, #sprite: 1]
  --
  -- If you know the sprite number of the list, then you can call it
directly:
  --
  --   put sendSprite (2, #DropList_Selection)
  --   -- [#item: 5, #text: "Main", #type: #movie, #sprite: 2]

  if stringP (propListOrString) then
    if propListOrString <> myName then exit
  end if

  data = ¬
[¬
 #item:   mySelectedItem, ¬
 #text:   myItemsList[mySelectedItem], ¬
 #type:   myContent, ¬
 #sprite: spriteNum ¬
]
  if ilk (propListOrString) <> #propList then
    return data
  else
    propListOrString.addProp(myName, data)
    return propListOrString
  end if
end DropList_Selection


on DropList_SetContents me, theContents, theListName
  -- Changes the contents of the dropdown list to theContents.  This can
be
  -- either a RETURN delimited string, a list, #marker or #movie.
  -- "theListName" is an optional parameter which allows you to change
the
  -- contents of a given list by using its name rather than by calling a
  -- specific sprite number or behavior.  Examples:
  --
  -- sendSprite (1, #DropList_SetContents, #movies)
  --
  -- sendAllSprites (#DropList_SetContents, #marker, "List 1")
  --
  -- objectRef = sendAllSprites (#DropList_GetReference, "Lingo")
  -- call (#DropList_SetContents, objectRef, ["Register","Help",
"Quit"])

  if not voidP (theListName) then
    if theListName <> myName then exit
  end if

  case ilk (theContents) of
    #string:
      myListMember.text = theContents
      myContent = #content
      Initialize me
    #list:
      listItems = ""
      lineCount = theContents.count
      repeat with i = 1 to lineCount
        theLine = string (theContents[i])
        put theLine&RETURN after listItems
      end repeat
      myListMember.text = listItems
      myContent = #content
      Initialize me
    #symbol:
      case theContents of
        #marker:
          myContent = #marker
          Initialize me
        #movie:
          myContent = #movie
          Initialize me
        otherwise
          return #invalidListContents
      end case
    otherwise
      return #invalidListContents
  end case
end DropList_SetContents


on DropList_ToggleExecution me, executeMode, listName
  -- Determines whether the dropdown list executes any lingo when an
item is
  -- selected, or whether it simply retains the selection data.
  --
  -- "listName" is an optional parameter.  You can use it to change the
  -- setting of a given list by using its name rather than by calling a
  -- specific sprite number or behavior.  This example makes the
dropdown list
  -- named "Movies" execute when an item is selected:
  --
  --    sendAllSprites (#DropList_ToggleExecution, TRUE, "Movies")
  --
  -- If you use a sendAllSprites message without the listName parameter,
all
  -- dropdown lists will be affected.  This example switches all
dropdown lists
  -- to #select mode:
  --
  --    sendAllSprites (#DropList_ToggleExecution, #select)
  --
  -- "executeMode" can take any of five values:
  -- * #execute and TRUE will make the dropdown list execute as
appropriate
  --    when an item is selected.
  -- * #select and FALSE will make the dropdown list do nothing when an
item
  --    is selected.  Use a #DropList_Selection call to get the
selection info.
  -- * void (or leaving the parameter blank) will toggle the behavior
between
  --    #execute and #select mode.
  -- You can use a list name as the only parameter; this will toggle the
  -- execute mode of the named list.  This example toggles the execute
mode of
  -- a dropdown list named "Lingo":
  --
  --    sendAllSprites (#DropList_ToggleExecution, "Lingo")
  --
  --  This example sets the dropdown list in sprite 1 to select mode:
  --
  --    sendSprite (1, #DropList_ToggleExecution, FALSE)


  if not voidP (listName) then
    if listName <> myName then
      exit
    end if
  else if stringP (executeMode) then
    -- Treat executeMode as if it were listName and executeMode were
void
    if executeMode = myName then
      myAction = not myAction
    else
      exit
    end if
  end if

  if voidP (executeMode) then
    myAction = not myAction
  else
    case executeMode of
      #execute, TRUE: myAction = TRUE
      #select, FALSE: myAction = FALSE
      otherwise
        return #invalidExecuteMode
    end case
  end if
end DropList_ToggleExecution


on DropList_GetReference me, propListOrString
  -- Returns the object reference of this behavior.  If you have several
  -- dropdown lists with different names, then you can use
sendAllSprites call
  -- with a property list as a parameter.  Example:
  --
  --   put sendAllSprites (#DropList_GetReference, [:])
  --   -- ["Markers": <objectRef>, "Movies": <objectRef>, "Lingo":
<objectRef>]
  --
  -- If you want the behavior reference of a particular list then use
its name:
  --
  --   put sendAllSprites (#DropList_GetReference, "Markers")
  --   -- <offspring "Dropdown List" 2 2ee89e8>
  --
  -- If you know the sprite number of the list, then you can call it
directly:
  --
  --   put sendSprite (1, #DropList_GetReference)
  --   -- <offspring "Dropdown List" 2 2ee89e8>

  case ilk(propListOrString) of
    #propList:
      propListOrString.addProp(myName, me)
      return propListOrString
    #string:
      if propListOrString = myName then return me
    otherwise
      return me
  end case
end DropList_GetReference



-- ERROR CHECKING --

on ErrorAlert me, theError, data
  -- sent by getPropertyDescriptionList, Initialize
  case theError of
    #getPDLError:
      alert "¬
Error: This behavior works only with Field members."&RETURN&RETURN&"¬
Hit OK and then delete this behavior from the sprite."&RETURN&RETURN&"¬
For more information on deleting Behaviors, see the Help system."
      if the optionDown then
        return ¬
[ ¬
 #getPDLError: ¬
 [ ¬
  #comment: "ERROR:   Wrong member type.   Click 'Cancel'."&RETURN&"¬
             Use only with Field members.", ¬
  #format:  #string, ¬
  #range:   [""], ¬
  #default:  "" ¬
 ] ¬
]
      end if

    #invalidMemberType:
      -- Determine the behavior's name
      behaviorName = string (me)
      delete word 1 of behaviorName
      delete the last word of behaviorName
      delete the last word of behaviorName

      alert "¬
BEHAVIOR ERROR: Frame "&the frame&", Sprite
"&me.spriteNum&RETURN&RETURN&"¬
Behavior "&behaviorName&" only works with Field members."&¬
RETURN&RETURN&"Current member type = #"&data
      halt
  end case
end ErrorAlert



-- AUTHOR-DEFINED PARAMETERS --

on getPropertyDescriptionList me

  if not the currentSpriteNum then exit

  -- Error check: does current sprite contain appropriate member type?
  theMember = sprite(the currentSpriteNum).member
  memberType = theMember.type
  permittedTypes = PermittedMemberTypes(me)
  if not permittedTypes.getPos(memberType) then
```

```
    return errorAlert (me, #getPDLError, permittedTypes)
  end if

  -- No errors detected: continue as usual ...

  return ¬
 [ ¬
  #myName: ¬
  [ ¬
   #comment: "Name of this list:", ¬
   #format:  #string, ¬
   #default: "List "&the currentSpriteNum ¬
  ], ¬
  #myContent: ¬
  [ ¬
   #comment: "Contents of list:", ¬
   #format:  #string, ¬
   #range:  ¬
   [¬
    "Current contents of the field", ¬
    "Markers in this movie", ¬
    "Movies with the same path name" ¬
   ], ¬
   #default: "Current contents of the field" ¬
  ], ¬
  #myAction: ¬
  [ ¬
   #comment: "Purpose of list:", ¬
   #format:  #string, ¬
   #range :  ¬
   [ ¬
    "Select:  return the selected item when called", ¬
    "Execute: go movie | go marker | do selectedLine" ¬
   ], ¬
   #default: "Select:  return the selected item when called" ¬
  ], ¬
  #myCheckmark: ¬
  [ ¬
   #comment: "Checkmark to indicate currently selected item:", ¬
   #format:  #string, ¬
   #default: ">" ¬
  ], ¬
  #myStandard: ¬
  [ ¬
   #comment: "Use standard style?", ¬
   #format:  #boolean, ¬
   #default:  TRUE ¬
  ] ¬
 ]
end getPropertyDescriptionList


on PermittedMemberTypes me
  -- sent by getBehaviorDescription, getPropertyDescriptionList,
  -- Initialize, ErrorAlert
  return [#field]
end PermittedMemberTypes

on initRegs
  initIRs
  initPC
  initStage2Regs
  initStage3Regs
  initStage4Regs
  initZ5
end


on initIRs
  global IR
  inst = [instType:8, op:"nop"]
  IR = []
  --load all 4 IRs with nop's
  repeat with i = 1 to 5
    add(IR, inst)
  end repeat
end

on loadRegisters
  loadPC

  --  loadStage2Regs
  loadStage4Regs
  loadStage3Regs

  --loadGenRegFile

  loadZ5





end


on initStage2Regs
  global stage2Regs, regFile
  regFile = [a1:"", R1:"", a2:"", R2:"", a3:"", R3:""]
  stage2Regs = [PC2:2004]
end


on initPC
  global PC
  PC = 2000
end


on loadPC
  global PC, switch

  PC = switch[#Mp1]

end


on loadIRs
  global IR, PC

  repeat with i = 5 down to 3
```

```
    IR[i] = IR[i - 1]
  end repeat

  IR[2] = getInstruction (PC)
  return

end

--gets instruction list for specified instMem index
on getInstruction (PC)
  global InstMemList, PC

  instFound = false
  --Search for instruction through inst address which is in the PC
  repeat with i = 1 to 15
    x = InstMemList[i][#instAddress]
    if InstMemList[i][#instAddress] = PC then
      instFound = true
      return InstMemList[i]
    end if

  end repeat

  alert("instruction out of bounds")
  go to frame "memoryDisplay"

  return
end




on loadStage2Regs
  loadPC2
  --loadGenRegFile
end


on initPC2
  global stage2Regs, logic

  stage2Regs[#PC2] = logic[#inc4]
end


on loadPC2
  global stage2Regs, logic

  stage2Regs[#PC2] = logic[#inc4]
end

on initStage3Regs
  global stage3Regs, switch
  stage3Regs = [X3:0, Y3:0, MD3:"R2 from RegFile"]
end

on loadStage3Regs
  global stage3Regs, switch
  stage3Regs = [X3:switch[#Mp3], Y3:switch[#Mp4], MD3:"R2 from RegFile"]
end
```

```
on initStage4Regs
  global stage4Regs, logic, stage3Regs

  stage4Regs = [Z4:0, MD4:0]
end


on loadStage4Regs
  global stage4Regs, logic, stage3Regs

  stage4Regs = [Z4:logic[#alu], MD4:stage3Regs[#MD3]]
end

on initZ5
  global Z5
  Z5 = 0
end

on loadZ5
  global Z5, switch

  Z5 = switch[#Mp5]

end


--controlCodeList = [branch:false, sh:false, alu:false, imm:false, ¬
load:false, ladr:false, store:false, ls:false, regWrite:false,
dsp:false, ¬
rl:false, cond:false]


on initSwitches
  global switch
  switch = [Mp1:2004, Mp2:0, Mp3:0, Mp4:0, Mp5:0]
end


on getSwitches

  getMp1
  getMp2
  getMp3
  getMp4
  getMp5
end


on getMp1
  global switch, controlSignals, logic, GENREG
  i = 2
  if controlSignals[i][#branch] and controlSignals[i][#cond] then
    switch[#Mp1] = GENREG[#R1]
  else
    switch[#Mp1] = logic[#inc4]
  end if
end


on getMp2
```

```
    global switch, controlSignals, logic, IR
    i = 2
    if controlSignals[i][#store] = true then
      switch[#Mp2] = IR[i][#ra]
    else
      switch[#Mp2] = IR[i][#rc]
    end if
  end

  on getMp3
    global switch, controlSignals, logic, stage2Regs, GENREG
    i = 2
    if controlSignals[i][#rl] or controlSignals[2][#branch] then
      switch[#Mp3] = stage2Regs[#PC2]
    else if controlSignals[i][#dsp] or controlSignals[2][#alu] then
      switch[#Mp3] = GENREG[#R1]
    end if
  end

  on getMp4
    global switch, controlSignals, IR, stage2Regs, GENREG
    i=2
    if controlSignals[i][#alu] and controlSignals[2][#imm] then
      switch[#Mp4] = GENREG[#R2]
    else if controlSignals[i][#dsp] or controlSignals[2][#imm] then
      switch[#Mp4] = IR[i][#c2]
    else if controlSignals[i][#rl] = true then
      switch[#Mp4] = IR[i][#c1]
    end if
  end

  on getMp5
    global switch, controlSignals, stage4Regs, MAINMEM
    i = 5
    if controlSignals[5][#load] = false then
      switch[#Mp5] = stage4Regs[#Z4]
    else if controlSignals[5][#load] = true then
      put char 1 to "m" of stage4Regs[#Z4]
      address = stage4Regs[#Z4]
      switch[#Mp5] = MAINMEM[address]
    else if controlSignals[5][#store] = true then
      put char 1 to "m" of stage4Regs[#Z4]
      MAINMEM[stage4Regs[#Z4]] = stage4Regs[#MD4]
    else if (controlSignals[5][#ladr] or controlSignals[5][#branch] or ¬
  controlSignals[5][#alu]) then
      switch[#Mp5] = stage4Regs[#Z4]
      loadMainMemFields
    end if


  end


  -- DESCRIPTION --

  on getBehaviorDescription me
    return "¬
  TOOLTIP" &RETURN&RETURN&"¬
  Generates a tool tip when the user rolls over the
  sprite."&RETURN&RETURN&"¬
```

```
NOTE: This behavior calls the 'Display Text' behavior to actually show
the ¬
message.  The 'Display Text' behavior must be attached to a different
sprite ¬
which contains either a Field or a Text member."&RETURN&RETURN&"¬
If such a sprite exists, it will automatically be selected in the
Behavior ¬
Parameters dialog."&RETURN&RETURN&"¬
If you wish the Tooltip to appear in a given position relative to the
current ¬
sprite, choose the appropriate position in the Behavior Parameters
dialog, and ¬
ensure that the associated 'Display Text' behavior is set to act as a ¬
Tooltip.  (If the 'Display Text' behavior is set to act as a Status Bar,
then ¬
it will ignore any position data and appear in a fixed position)."&¬
RETURN&RETURN&"¬
You can choose to have the tooltip appear immediately on rollover, or to
¬
appear only if the mouse remains over the sprite for a given period.
You can ¬
also choose to have the tooltip disappear if the user clicks on the
sprite."&¬
RETURN&RETURN&"¬
The Behavior Parameters dialog has limited space for entering a tool tip
¬
message.  In particular, it will not accept a string which contains the
RETURN ¬
character.  If you need to display a long Tooltip which consists of
several ¬
lines of text, and which must appear at the position of this sprite,
then you ¬
must use send a message to this behavior containing the requiresd
string.  ¬
For example:"&RETURN&RETURN&"¬
  SendSprite (1, #Tooltip_SetMessage, "&QUOTE&"This message
consists"&QUOTE&"¬
&RETURN&"&QUOTE&"of two lines of text"&QUOTE&")"&RETURN&RETURN&"¬
This would produce the following message when the mouse rolls over
sprite 1:"&¬
RETURN&RETURN&"¬
     This message consists"&RETURN&"¬
     of two lines of text"&RETURN&RETURN&"¬
If the tooltip generated by this behavior is to be diplayed in a Status
bar ¬
then this step may be needed.  The 'Display Text' behavior will ensure ¬
that a long line of text is wrapped in the Status bar, and that scroll
bars ¬
appear if necessary."&RETURN&RETURN&"¬
PERMITTED MEMBER TYPES:"&RETURN&"All"&RETURN&RETURN&"¬
PARAMETERS:"&RETURN&"¬
* Tool tip to display (single-line string)"&RETURN&"¬
* Delay before displaying tool tip (0 - 2 seconds)"&RETURN&"¬
* Hide tool tip if sprite is clicked? (TRUE | FALSE)"&RETURN&"¬
* Position of tool tip relative to the sprite"&RETURN&"¬
  (This will be ignored if the 'Display Text' behavior ¬
is set to act as a status bar)."&RETURN&"¬
* Number of the sprite where tool tip is to be displayed."&RETURN&"¬
  (This sprite should have the 'Display Text' behavior attached to it.
If the ¬
given sprite is moved an authortime alert will invite you to update the
¬
Behavior Parameters)."&RETURN&RETURN&"¬
PUBLIC METHODS:"&RETURN&"¬
=> Set the tooltip message (allows the RETURN character)"&RETURN&"¬
=> Obtain behavior reference"&RETURN&RETURN&"¬
ASSOCIATED BEHAVIORS:"&RETURN&"¬
+ Display Text - ESSENTIAL - must be attached to a Field or Text sprite
which ¬
covers the same span of frames."&RETURN&RETURN&"¬
You can find the 'Display Text' behavior in the Library Palette, under ¬
Media > Text > Display Text."
end getBehaviorDescription


on getBehaviorTooltip me
  return "¬
Use with any type of member to generate a tool tip"&RETURN&"¬
message when the mouse is over the sprite"&RETURN&RETURN&"¬
This behavior requires that the 'Display Text' behavior"&RETURN&"¬
be available on a Field or Text sprite to display the"&RETURN&"¬
messages that it generates.  If no such sprite is"&RETURN&"¬
available an alert will appear (authortime only)."&RETURN&RETURN&"¬
The parameter set for the associated 'Display Text'"&RETURN&"¬
behavior determines whether the tool tip message"&RETURN&"¬
appears in a Status Bar or as a temporary Tooltip"&RETURN&"¬
display over or near this sprite."
end getBehaviorTooltip



-- NOTES FOR DEVELOPERS --

-- The tool tip member appears in a separate sprite: myDisplaySprite.
This
-- behavior merely tells the 'Display Text' behavior on myDisplaySprite
what
-- text to display.  Before it can do this, it must first discover the
memory
-- address where the 'Display Text' behavior is stored.  It does this
through
-- the  EnrollDisplaySprite handler.  This is not called on beginSprite
-- since, logically, myDisplaySprite will be in a higher channel.  If
both
-- sprites begin in the same frame, myDisplaySprite may not have been
-- initialised yet.

-- You may have more than one sprite with the 'Display Text' behavior.
It may
-- therefore be important to ensure that the Tooltip is sent to the
right
-- sprite.  However, while authoring, sprites tend to get moved around.
The
-- EnrollDisplaySprite does its best to find a new instance of the
-- 'Display Text' behavior, but it warns the author that a conflict may
be
-- occurring.
--
-- DECIDING WHEN TO DISPLAY THE TOOL TIP
-- The tooltip should only appear when the mouse is over the current
sprite.
```

```
-- You may want the it only to be visible when the mouse is up.  The
-- CheckStatus handler, called on each prepareFrame analyses the
position and
-- state of the mouse, and either calls ShowTip, HideTip, or leaves
things as
-- they are.
--
-- HideTip doesn't in fact hide anything.  It merely displays an empty
string.
-- If the 'Display Text' behavior is set to act as a Tooltip, it will
move its
-- sprite off-stage.  If it is set to act as a Status Bar, it will
simply empty.
--
-- Since many behaviors may be sending messages to myDisplaySprite, I
make sure
-- that these messages don't conflict.  To do this, I use a boolean
property:
-- myDisplayFlag.  This toggles between TRUE and FALSE.  While it is
TRUE, no
-- further ShowTip messages are generated.  It is set to FALSE when the
tool
-- tip is hidden: HideTip is thus also only called once.
--
-- I could have chosen to use mouseEnter and mouseLeave to show and hide
the
-- tip.  This technique would make the tool tip appear immediately.  I
chose to
-- allow up to 2 seconds delay (120 ticks) before showing the tip.
Instead of
-- using mouseEnter to trigger the ShowTip call directly, I use it to
-- calculate the value of the ticks when the call should be make.  I
store the
-- result in a property: myStartTicks.  By setting myStartTicks to
-- the maxInteger on mouseLeave, I can be sure that ShowTip will not be
-- triggered.
--
-- POSITION OF TOOL TIP
-- ShowTip calculates where the tool tip should (ideally) be shown.  The
-- 'Display Text' behavior on myDisplaySprite may overrule this for one
of two
-- reasons:
--   * You have set the 'Display Text' behavior to act as a Status bar
--   * The ideal position of the tool tip would place it (partially) off-
stage.
--
-- See the 'Notes for Developers' in the 'Display Text' behavior for
more
-- details.


-- HISTORY --

-- 1 October 1998:  written for the D7 Behaviors Palette by James Newton
-- 29 October 1998: descriptions improved, myPosition property extended.


-- PROPERTIES --

property mySprite
```

```
-- error checking
property getPDLError
-- author-defined parameters
property myString
property myDelay
property myPosition
property myDisplaySprite
property myHideFlag
-- internal properties
property myDisplayList
property myStartTicks
property myDisplayFlag


-- EVENT HANDLERS --

on beginSprite me
  Initialize me
end beginSprite


on prepareFrame me
  CheckStatus me
end prepareFrame


on mouseEnter me
  myStartTicks = the ticks + myDelay
end mouseEnter


on mouseLeave me
  myStartTicks = the maxinteger
end mouseLeave



-- CUSTOM HANDLERS --

on Initialize me -- sent by beginSprite
  mySprite = sprite(me.spriteNum)
  myMember = mySprite.member
  -- Error checking
  if not voidP (getPDLError) then
    ErrorAlert (me, #getPDL_Invalid, myMember.type)
  end if
  -- End of error checking

  myDisplayList = []
  myStartTicks = the maxinteger
end Initialize


on CheckStatus me
  if myStartTicks < the ticks  then
    if myHideFlag then
      if the mouseDown then
        if myDisplayFlag then
          HideTip me
        end if
        exit
      end if
    end if

    if myDisplayFlag then exit

    ShowTip me
  else if myDisplayFlag then
    HideTip me
  end if
end CheckStatus


on ShowTip me -- sent by prepareFrame
  myDisplayFlag = TRUE
  case myPosition of
    "centered above":
      theAlignment = #bottomCenter
      displayLoc = point ((mySprite.left + mySprite.right) / 2,
mySprite.top)
    "centered below":
      displayLoc = point ((mySprite.left + mySprite.right) / 2,
mySprite.bottom)
      theAlignment = #topCenter
    "at topLeft":
      displayLoc = point (mySprite.left, mySprite.top)
      theAlignment = #topLeft
    "at topRight":
      displayLoc = point (mySprite.right, mySprite.top)
      theAlignment = #topRight
    "centered":
      centerH = (mySprite.left + mySprite.right) / 2
      centerV = (mySprite.top + mySprite.bottom) / 2
      displayLoc = point (centerH, centerV)
      theAlignment = #center
    "at bottomLeft":
      displayLoc = point (mySprite.left, mySprite.bottom)
      theAlignment = #bottomLeft
    "at bottomRight":
      displayLoc = point (mySprite.right, mySprite.bottom)
      theAlignment = #bottomRight
    "at regPoint":
      displayLoc = mySprite.loc
      theAlignment = #center
    "under the mouse":
      displayLoc = the mouseLoc
      theAlignment = #center
    #bottomLeft: displayLoc = point (mySprite.left, mySprite.bottom)
  end case

  if not myDisplayList.count() then
    EnrollDisplaySprite me
  end if
  call #DisplayText_SetText, myDisplayList, myString, displayLoc,
theAlignment
end ShowTip


on HideTip me -- sent by prepareFrame
  myDisplayFlag = FALSE
  if not myDisplayList.count() then
    EnrollDisplaySprite me
  end if
  call #DisplayText_SetText, myDisplayList, EMPTY
end HideTip


on EnrollDisplaySprite me
  -- Enroll the 'Display Text' behavior
  sendSprite (myDisplaySprite, #DisplayText_Enroll, myDisplayList)
  if not myDisplayList.count() then
    -- Try to find a sprite with the 'Display Text' behavior anyway
    sendAllSprites (#DisplayText_Enroll, myDisplayList)
    if not myDisplayList.count() then
      ErrorAlert (me, #noValidSprites, myDisplaySprite)
    else
      -- Notify author of change
      ErrorAlert (me, #invalidSpriteNumber, myDisplaySprite)
    end if
  end if
end EnrollDisplaySprite


on GetDisplaySprite me
  -- Checks the scriptList of each sprite in this frame for 'Display
Text'
  displayScriptMember = the number of member ("Display Text")
  if displayScriptMember > 0 then
    displayScriptMember = member (displayScriptMember)
    repeat with theSprite = 1 to the lastChannel
      theScripts = sprite (theSprite).scriptList
      scriptCount = theScripts.count()
      repeat while scriptCount
        if theScripts[scriptCount][1] = displayScriptMember then
          return theSprite
        end if
        scriptCount = scriptCount - 1
      end repeat
    end repeat
  end if
  return the currentSpriteNum + 1
end GetDisplaySprite


-- PUBLIC METHODS (responses to #sendSprite, #sendAllSprites, #call) --

on Tooltip_SetMessage me, theString
  -- Allows you to set a more complex message that can be done through
the
  -- Behavior Parameters Dialog, or to change it at runtime

  -- Error check
  case ilk (theString) of
    #string: -- nothing
    otherwise
      return #invalidTypeError
  end case
  -- End of error check

  myString = theString
end Tooltip_SetMessage
```

```
on Tooltip_GetReference me
  -- Returns a reference to the behavior for Lingo calls
  return me
end Tooltip_GetReference



-- ERROR CHECKING --

on ErrorAlert me, theError, data
  -- sent by getPropertyDescriptionList, Initialize
  -- Determine the behavior's name
  behaviorName = string (me)
  delete word 1 of behaviorName
  delete the last word of behaviorName
  delete the last word of behaviorName
  -- Convert #data to useful value
  case data.ilk of
    #void: data = "<void>"
    #symbol: data = "#"&data
  end case

  case theError of
    #invalidSpriteNumber:
      if the runMode = "Author" then
        alert"¬
BEHAVIOR ERROR: Frame "&the frame&", Sprite "&me.spriteNum&RETURN&"¬
Behavior "&behaviorName&&RETURN&RETURN&"¬
Sprite "&data&" did not respond to a #DisplayText call.  Another sprite
will ¬
be used.  Please open the Behavior Parameters dialog to choose ¬
the correct sprite for displaying the Tooltip message."
      end if

    #noValidSprites:
      if the runMode = "Author" then
        alert "¬
BEHAVIOR ERROR: Frame "&the frame&RETURN&"¬
Behavior "&behaviorName&&RETURN&RETURN&"¬
No sprites responded to a #DisplayText call."&RETURN&RETURN&"¬
Please ensure that the 'Display Text' behavior is attached to a Field or
Text¬
Sprite in the same frames as Sprite "&me.spriteNum
      end if

  end case
end ErrorAlert



-- AUTHOR-DEFINED PARAMETERS --

on getPropertyDescriptionList me

  -- try to find a sprite which has the 'Display Text' behavior attached
  displaySprite = GetDisplaySprite (me)

  return ¬
[ ¬
 #myString: ¬
```

```
  [ ¬
   #comment:  "Text of tool tip:", ¬
   #format:   #string, ¬
    #default: "Insert your single-line tool tip here" ¬
  ], ¬
  #myDelay: ¬
  [ ¬
   #comment: "Pause before showing tool tip (ticks):", ¬
   #format:  #integer, ¬
   #range:  [#min: 0, #max: 120], ¬
   #default:  30 ¬
  ], ¬
  #myHideFlag: ¬
  [ ¬
   #comment: "Hide tool tip if user clicks on sprite?", ¬
   #format:  #boolean, ¬
   #default:  TRUE ¬
  ], ¬
  #myPosition: ¬
  [ ¬
   #comment: "Tool tip position relative to sprite (see notes):", ¬
   #format:  #string, ¬
   #range:  ¬
   [ ¬
    "centered above", ¬
    "at topLeft", ¬
    "at topRight", ¬
    "centered", ¬
    "at bottomLeft", ¬
    "at bottomRight", ¬
    "centered below", ¬
    "at regPoint", ¬
    "under the mouse" ¬
   ], ¬
   #default: "centered" ¬
  ], ¬
  #myDisplaySprite: ¬
  [ ¬
   #comment: "Use which sprite to display tooltip?", ¬
   #format:  #integer, ¬
   #range:   [#min: 1, #max: the lastChannel], ¬
   #default:  displaySprite ¬
  ] ¬
 ]
end getPropertyDescriptionList
```

LIST OF REFERENCES

[1] Gayeski, Diane M., *Designing and Managing Computer Mediated Learning: An Interactive Toolkit*, OmniCom Associates, Ithaca, NY 14850, 1997.

[2] Fluckinger, Francois, *Understanding Networked Multimedia Applications and Technologies*, Prentice Hall, NY, 1995.

[3] Ravet, Serge and Layte, Maureen, *Technology-Based Training*, Gulf Publishing Company, Houston, TX, 1997.

[4] Ruth Colvin Clark, Ph.D,"Authorware, Multimedia, and Instructional Methods", URL:http//www.macromedia.com/support/authorware/attain/how/expert/instruct/instruct.html, 1998.

[5] Boyle, Tom, *Design for Multimedia Learning*, Prentice Hall, NY, 1997.

[6] Reynolds, Angus and Iwinski, Thomas, *Multimedia Training, Developing Technology Based Systems*, McGraw Hill, NY, 1995.

[7] Macromedia Support Team, Director 7 Internet Studio Product information, http://www.macromedia.com/software/director/productinfo, 1998

[8] Plant, Darrel, *Flash 3! Creative Web Animation*, Macromedia Press, CA, 1998.

[9] Macromedia Support Team, Fireworks, http://www.macromedia.com/software/fireworks/productinfo/, 1998.

[10] Towers, Tarin J., *Dreamweaver for Windows and Macintosh*, Peachpit Press, Berkeley, CA, 1998.

[11] Tauber, Daniel A., *Mastering Microsoft FrontPage 98*, Sybex, CA, 1998.

[12] Kellog, Orson, *Macromedia Authorware 5 Attain Authorized*, Macromedia Press, CA, 1998.

[13] Kirkpatrick, Donald, *Evaluating Training Programs: The Four Levels*, Berret-Koeher, NY, 1994.

[14] Lynch, P. J. and Horton, S., *Web Style Guide*, Yale University Press, New Haven, CT, 1999.

[15] Heuring, V., Jordan, H., *Computer Systems Design and Architecture,* Addison Wesley, CA, 1997.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center........................2
   8725 John H. Kingman Rd., STE 0944
   Ft. Belvoir, Virginia 22060-6218

2. Dudley Knox Library........................2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, California 93943-5000

3. Chairman, Coded EC........................1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California 93943-5121

4. Professor Jon T. Butler, Coded EC/Bu........................1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California 93943-5121

5. Professor Hersch Loomis, Coded EC/He........................1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California 93943-5121

6. Professor Herschel Loomis, Coded EC/Lm........................1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California 93943-5121

7. Professor John McEachen, Coded EC/Mj........................1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California 93943-5121

8. Professor Murali Tummala, Coded EC/Tu........................1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California 93943-5121

9. Professor Alene Guest, Coded EC/Gt........................1
   Department of Oceanography
   Naval Postgraduate School
   Monterey, California 93943-2130

10. Tom Hazard, Coded IDEA/Cr..1
    Criscoulo Hall
    Naval Postgraduate School
    Monterey, California 93943-3110

11. LT Dave Floodeen, Coded 32..1
    Department of Electrical and Computer Engineering
    Naval Postgraduate School
    Monterey, California 93943-5121

12. Darren S. Harvey..1
    3011 Pearce St.
    Jacksonville, Florida 32209

13. LCDR Brian Murphy, Code 054-22..1
    SPAWAR
    4301 Pacific Highway
    San Diego, California 92111